Design of a PLC Control Program for a Batch Plant VHS Case Study 1

Angelika Mader[1]*, Ed Brinksma[2], Hanno Wupper[1], Nanette Bauer[3]
[1]Computer Science Department, University of Nijmegen
[2]Faculty of Computer Science, University of Twente
[3]Department of Chemical Engineering, University of Dortmund

April 5, 2001

Abstract

This article reports on the systematic design and validation of a PLC control program for the batch plant that has been selected as a case study for the EC project on Verification of Hybrid Systems (VHS). We show how a correct design of the control program can be obtained in an incremental manner using a real-time logical formalism. This is done by systematically strengthening the premise of an implication whose conclusion represents the required behaviour of the plant. The premise specifies the assumptions under which this behaviour is realised. The formal proof of correctness was obtained using formal verification tools. We used both theorem-proving (PVS) and model checking (Spin) as verification strategies. With PVS we could show the correctness of the final implication directly by a semantic embedding of the real-time logic in PVS, but only for a limited operational scenario (a single batch load). With Spin we could show the correctness for all relevant operational scenarios, but only indirectly, viz. on the basis of an abstract verification model (written in Promela). This model was obtained as a straightforward translation of the premise of the final version of the formal design and the PLC code derived from it. We conclude that the judicious use of standard formal methods and tools suffices for the systematic development of correct control programmes for this kind of application.

Keywords: *Hybrid Systems, Plant Control, Specification Method, Theorem Proving, Model Checking*

1 Introduction

In the past decade the study of hybrid systems has proved itself a fertile topic for researchers from computer science and systems and control theory alike. The EU LTR project on the *Verification of Hybrid Systems* (VHS) [24] has brought together researchers from both communities to study the use of existing and development of new techniques for showing the correctness of systems that involve the combination of both discrete and continuous phenomena. As part of the project a number of case studies concerning particular examples of hybrid systems are being elaborated. This article reports on our experience with VHS case study 1 and the results we obtained.

*email: mader@cs.kun.nl, supported by an NWO postdoc grant, and the EC LTR project VHS (project nr. 26270)

Our approach to VHS case study 1 deals with the design and verification of a control program for a so-called batch plant that is controlled by a Programmable Logic Controller (PLC). A detailed description can be found in [11]. We recapitulate the main facts:

The batch plant (see figure 1) of the case study is an experimental plant, originally designed for student exercises. It "produces" batches of diluted salt solution from a concentrated salt solution (in container B1) and water (in container B2). These ingredients are mixed in container B3 to obtain the diluted solution, which is subsequently transported to container B4 and then further on to B5. In container B5 an evaporation process is started. The evaporated water goes via a condenser to container B6, where it is cooled and pumped back to B2. The remaining hot, concentrated salt solution in B5 is transported to B7, cooled down and then pumped back to B1.

The controlled batch plant is clearly a hybrid system. The discrete element is provided by the control program and the (abstract) states of the valves, mixer, heater and coolers (open/closed, on/off). Continuous aspects are tank filling levels, temperatures, and time. The latter can be dissected into real-time phenomena of the plant on the one hand, such as tank filling, evaporation, mixing, heating and cooling times times, and the program execution and reaction times (PLC scan cycle time), on the other. The controller of the batch plant is a nice example of an *embedded system*: the controlling, digital device is part of a larger physical system with a particular functionality.

The described batch plant has been used by many authors to illustrate their methods for the modelling and analysis of this kind of hybrid systems, see e.g. [21, 15, 19]. In section 6 we compare our work with that of others in the light of our results. In our elaboration of this case study we do not conduct an a posteriori verification of a given control program, but want to see it as an excercise in the systematic design of a correct controller. This has the following reasons:

- Following a structured design method helps us to obtain a nicely structured control program, whose structure can be exploited in the subsequent verification.
- From an engineering point of view it is interesting in its own right to see to what extent we can arrive at a generic design method for controllers for this kind of embedded systems. The availability of such methods can motivate engineers to make more use of formal methods and tools to obtain correct controllers.

Typically, embedded systems consist of two complementary parts: the environment to be controlled – the plant – and the controller. The desired behaviour of an embedded system concerns the joint behaviour of those part, i.e. the behaviour of the whole, controlled system. The design and verification process, therefore, must take both parts into account. As the result of the design process we want to have a detailed specification of the controller. From the above considerations it follows that such a design process must somehow also include a specification of all relevant properties of the plant. For industrial-size projects this constitutes a complicated, labour-intensive and therefore error-prone task. In this paper we demonstrate a method that helps to govern this design process in a systematic and straightforward way.

We show how a correct design of the control program can be obtained in an incremental manner using a real-time logical formalism. This is done by systematically strengthening the premise of an implication whose conclusion represents the required behaviour of the plant. The premise specifies the assumptions under which this behaviour is realised. Our design procedure strengthens the premise in a step by step fashion, reflecting the accumulated design decisions, until the premise yields a sufficiently detailed description of the required program and the implication is true.

To prove the correctness of the design different techniques and tools can be used. One is to address the question on the abstraction level of the specification, and provide a logical proof of the implication, which entails that the specified plant and the controller together imply the desired

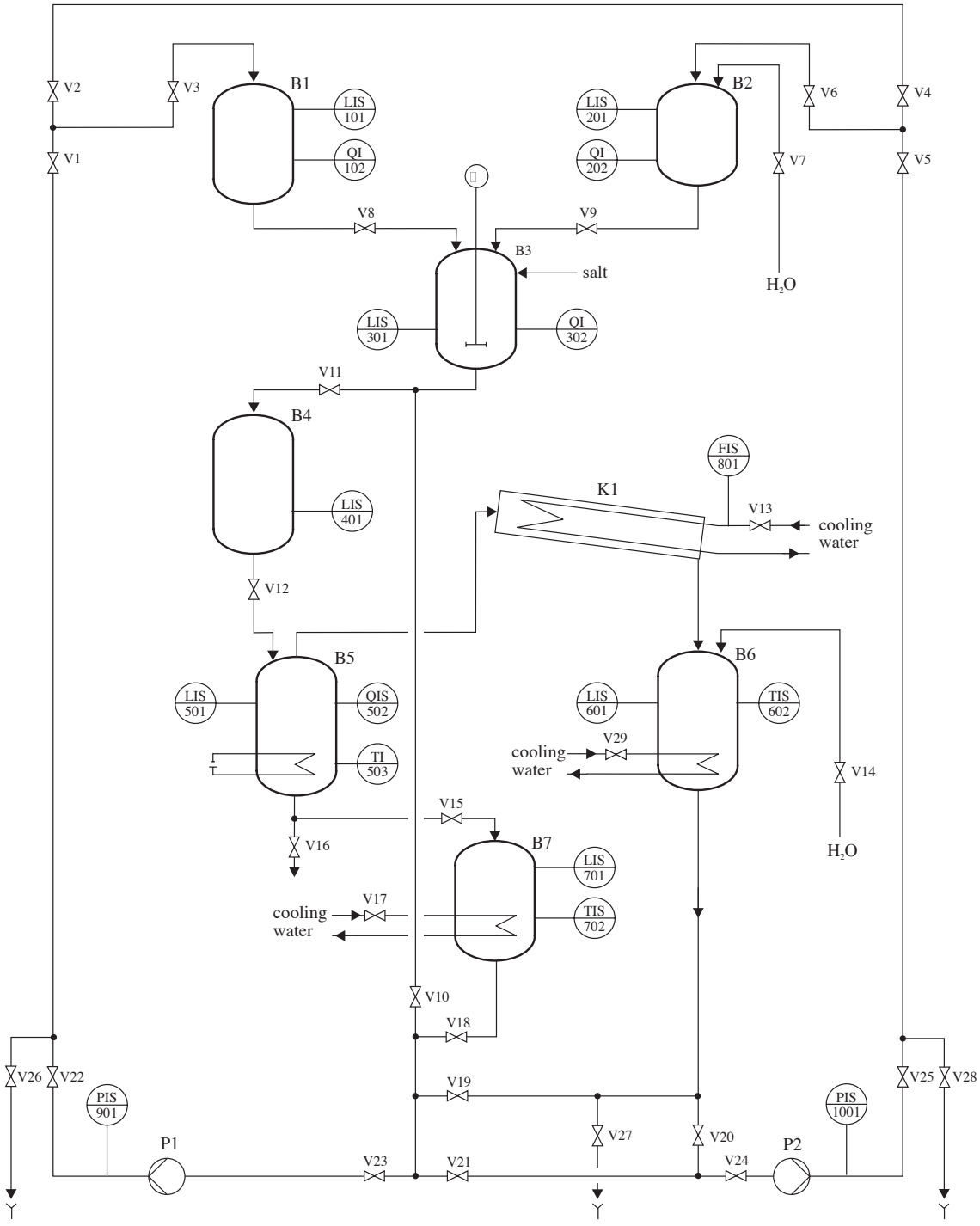

Figure 1: The P/I-Diagram of the Batch Plant

behaviour. For a simple instance of the case study where only a single batch is produced we have done this with help of PVS. For the more general case of multiple batches, however, this leads to an explosion of proof obligations that make a straightforward proof practically unmanageable (at least in our set-up), which motivated us to use the model-checking tool Spin [8] instead. The required Promela model was obtained as a straightforward translation of the ultimate version of the implication premise and the PLC code derived from it.

Our approach can be seen as an attempt to use the case study to assess how much of existing theory and tools for non-hybrid systems can be put to effective use in the case of plant-control type applications, rather than a testing ground for more specialized, advanced methods. In our elaboration we only work with batches of fixed sizes, which is a reasonable assumption that allows us to abstract from one of the continuous parameters. As we will show, the control program can be designed independently of the time that the individual processing steps of the plant take. This also simplifies the verification task considerably, as the correctness of the untimed case implies the correctness of the timed one. As the production steps in the plant are executed in parallel using shared resources, our solution involves an ad hoc scheduling policy to ensure mutual exclusion. We will not address the question of optimal scheduling, but indicate the extent to which our approach can be used for this purpose in the conclusions.

The organisation of the article is as follows: section 2 introduces the design method that we use, followed by its application in section 3; section 4 proceeds to give the implementation of the control program, whose correctness is then verified in section 5. In section 6 we compare our work to that of others. Section 7, finally, contains our conclusions.

2 The design method

2.1 A framework for incremental design

As we wrote in the introduction we intend to design a correct control program for the batch plant of case study 1. As we want to show the correctness of this design we want to conduct its development within a suitable formal framework. As often with complex designs, it will be a good idea to follow an incremental strategy, in which successive design steps accumulate into a final design that specifies the required control program in sufficient detail for straightforward implementation.

The strategy that we follow is based on the approach proposed in [27], of which we describe here the elements that we will need for the purpose of this article. This approach offers a framework for the notation and administration of the succesive design steps in terms simple logical properties and proof obligations. The idea is that the ultimate design is obtained as the extra assumptions with which the properties of the plant must be strengthened to imply the required behaviour.

The framework can be combined with different logical formalisms as long as they include the usual propositional operators and, of course, are rich enough to describe the relevant properties for the design at hand. It is built around two well-known principles (we present them here in their propositional format; in the design later we will use a slightly richer form including some modal operators):

- *divide and conquer* : as usual we will want to break up a complicated specification S into a combination of simpler ones, say $s_1,\ldots,s_n$, with the corresponding obligation that $\bigwedge_{1\leq i\leq n} s_i \Rightarrow S$.
- *assumption and commitment* : we will work with specifications S of the form $A \Rightarrow C$,

where C denotes the properties that a (sub)system must fulfil if the assumptions A about its environment hold.

The combination on these two structuring principles leads to the following canonical form at that can be used to specify the correctness requirement at any stage of the design:

$$\bigwedge_{1\leq i\leq n} (a_i \Rightarrow c_i) \Rightarrow (A \Rightarrow C) \tag{1}$$

Here, assumptions and commitments of the subsystems are denoted using lower case variables. Usually, this implication is not a logical tautology and can be proven only relative to a suitable additional theory of extra-logical principles, formalising domain-specific knowledge, natural laws, etc.

When using formal methods to support design it is important to distinguish between the *context of discovery*, i.e. the actual process of design as a creative activity, and the *context of justification*, i.e. the a posteriori justification of the final design as the outcome of a rigorously applied formal procedure. Formal methods are mostly presented in the context of such a formal justification. The syntactic format of (1), however, can be used to guide actual design, viz. by analysing what to do when at a given design stage the proof of (1) is seen to fail. The logical form suggests the following possible cures:

- weakening C, i.e. designing a system that meets weaker overall requirements;
- strengthening A, i.e. making stronger assumptions about the overall environment;
- strengthening c_i, i.e. designing components that meet stronger requirements;
- weakening a_i, i.e. designing components that can operate in less co-operative environments.

We can apply these principles to a specification of the plant behaviour to tease out a specification of the required control behaviour. The idea is, roughly speaking, as follows:

1. Analysing the overall production process of the plant we see that it is composed of a number of dedicated *production processes* satisfying some subsystem specifications $a_i \Rightarrow c_i$, and collectively fulfilling a suitable instantiation of (1).
2. Using our knowledge of the *plant*, we can see that its parts do not implement the requirements $a_i \Rightarrow c_i$, but the weaker implications $a_i \wedge p_i \Rightarrow c_i$, where the p_i represent the "control fragments" that are needed to operate the parts.
3. Replacing the $a_i \Rightarrow c_i$ in (1) by $a_i \wedge p_i \Rightarrow c_i$ will generally not yield a true theorem. Simple logical manipulation, however, shows that if (1) holds then the following implication also holds:

$$\bigwedge_{1\leq i\leq n} (a_i \wedge p_i \Rightarrow c_i) \wedge \bigwedge_{1\leq i\leq n} (a_i \Rightarrow p_i) \Rightarrow (A \Rightarrow C)$$

4. Taking $\bigwedge_{1\leq i\leq n}(a_i \Rightarrow p_i)$ as the first specification of the control program, we may (in fact, do) find that it is physically unimplementable – the necessary coordination between the control fragments is still missing – which motivates a further weakening of the control specification, etc.

We use the framework to structure and document the design decisions for a control program for the plant, not for a completely formal derivation of this program. In fact, we will not carry out any formal (dis)proofs of the intermediate implications that will be produced as part of the design process, but rely on insight in logical and engineering principles to assess the truth or falsehood of these statements. This will be feasible for all but the last stages of the design. The outcome of this process is a completely formal statement of the final design with a structured history of design decisions. To show the formal correctness of the final design we verify the resulting specification theorem with the aid of verification tools. The PVS proof that we carried out for one particular case replays formally many of the semi-formal design steps, of course.

In section 6 on related work we will contrast our approach with other work on formally supported incremental design for real-time systems [16, 3].

2.2 The specification formalism

For this paper we want a specification formalism which does not confine our results to any particular specification language, yields clear specifications for our case study, and can be easily translated into whatever formalism might later turn out to be advantageous.

Our first choice was (typed) predicate logic. It turned out that the specification elements could naturally be expressed in the following format, which relates three predicates *precondition, invariant* and *postcondition* in always the same way:

$$\begin{aligned}\forall\, t : \text{TIME}\ \exists\ \text{duration} : \text{TIME}\\ (\ \text{precondition}(t) \wedge (\forall t' : [0, \text{duration}).\ \text{invariant}(t + t'))\\ \Rightarrow \text{postcondition}(t + \text{duration}))\end{aligned} \tag{2}$$

Such specifications, however, are clumsy and error-prone, due to the many quantified time variables. Readability can be improved by suitable model operators inspired by [13]. This leads to a simple modal real-time formalism with straightforward translations to other real-time formalisms.

First, we give an informal introduction to the formalism.

Formulas are built from *atomic observations*. These are statements about the values of variables that represent the state of the physical system, the so called *observation points*. The observation points of our system are: the valves V_1, V_2, ..., the containers B_1, B_2, ..., the heater H_1, the mixer M_1 and the pumps P_1 and P_2. At each of these points at each moment one of a finite number of values can be observed. The valves can be open or closed, the heater can be on or off, while for each container a finite number of different contents, concentrations, temperatures, and filling levels can be distinguished. We simply write $V_{12} = \text{open}$ to specify that valve V12 can be observed to be open at the moment under consideration ('*now*').

Propositions of the language are now built recursively using:

- the propositional connectives: $\wedge$ (*and*), $\vee$ (*or*), $\Rightarrow$ (*implies*)
- the time shift operator: $\bigcirc^t$ (*at time t from now*)
- the bounded modal quantifier $\Box^t$: (*always up to time t from now*)
- the unbounded modal quantifier $\Box$: (*at any time from now*)
- the bounded modal quantifier $\Diamond^t$: (*eventually before time t from now*)

- the unbounded modal quantifier $\Diamond$: (*eventually*)

In this language we can write statements like:

$$\Box\ (B_1 = \text{empty} \wedge\ \Box^{d_{max}}\ V_{12} = \text{closed} \Rightarrow\ \bigcirc^{d_{max}} B_1 = \text{empty})$$

with the intuitive meaning: "at any time u (from now) it holds that if B_1 is empty at u and V_{12} is closed throughout the half-open interval $[u, u + d_{max})$, then B_1 will be empty at $u + d_{max}$".

More formally, formulas are interpreted over timed sequences of states of the observation points. Such sequences characterise potential behaviour of the system.

The formal definition of the syntax and semantics are as follows. Let $\mathcal{O}$ be a finite set of observation points, TIME be modelled as real numbers, and $\mathcal{V}$ be the set of observable values. At each observation point, at each moment a value can be observed. Our model thus is $\mathcal{S} =_{df} \mathcal{O} \to (\text{TIME} \to \mathcal{V})$.

Specifications are interpreted with respect to a model $s \in \mathcal{S}$ and a moment of reference $t \in \text{TIME}$. In the sequel, P and Q denote specifications, d and x points in time, $P \in \mathcal{O}$ an observation point and $v \in \mathcal{V}$ a value.

$s,t \models p = v$	iff	$s(p)(t) = v$
$s,t \models p = (v_1 \vee\ v_2)$	iff	$s(p)(t) = v_1$ or $s(p)(t) = v_2$
$s,t \models P \wedge\ Q$	iff	$s,t \models P$ and $s,t \models Q$
$s,t \models P \vee\ Q$	iff	$s,t \models P$ or $s,t \models Q$
$s,t \models P \Rightarrow\ Q$	iff	$s,t \models P$ implies $s,t \models Q$
$s,t \models \bigcirc^d P$	iff	$s,t + d \models P$
$s,t \models \Box^d\ P$	iff	$s,t + x \models P$ for all $x \in [0, d)$
$s,t \models \Box\ P$	iff	$s,t + x \models P$ for all $x \geq 0$
$s,t \models \Diamond^d\ P$	iff	$s,t + x \models P$ for at least one $x \in [0, d)$
$s,t \models \Diamond\ P$	iff	$s,t + x \models P$ for at least one $x \geq 0$
$s,t \models P\ \mathit{Until}\ Q$	iff	there exists a $u \geq 0$ with
		$s,t + x \models P$ for all $0 \leq x < u$ and $s,t + u \models Q$

With these operators the standard form (2) from above can be expressed more succinctly:

$$\Box\ (\text{precondition} \Rightarrow\ (\Box\ \ \text{invariant} \Rightarrow\ \Diamond\ \ \text{postcondition})) \tag{3}$$

The intuition for this presentation of the property is: if the precondition holds initially, then it cannot be the case the the invariant holds infinitely long, but the postcondition never will hold.

2.3 Tool support

We chose the proof assistent PVS [22] not only as one of the ways to support verification, but also as a tool to support the design process. We started from a collection of PVS theories [28] where the operators of our formalism are embedded in a more general framework and the usual axioms of temporal logic are available as theorems. We instantiated this framework to obtain our particular specification formalism and tailored it to our need to model chemical plants with their tanks and valves.

Even if one does not strive for formal verification by theorem proving the use of PVS as a design support tool has a number of advantages. The PVS language prides the usual facilities to structure complex specifications by defining notational abbreviations, introducing names for subformulas, etc. Moreover, if good use is made of the PVS typing system during language definition, the PVS type checker is a valuable tool to arrive at syntactically consistent specifications and rule out many

errors. The PVS module mechanism allows to formulate generic concepts like a general production step and instantiate these with, in our case, specific tanks, valves, and durations. In contrast to many popular design tools, everything is completely formal and well-defined. A specification obtained in this way is a suitable starting point for formal verification. Finally, the approach is suitable to obtain a "rapid prototype" of the specification formalism, which can be used to experiment with and debug specifications. PVS was indeed used to this effect to develop our first specifications.

A disadvantage of the PVS language is the readability of its ASCII-notation. In this article we represent all specifications in a more readable usual notation as introduced in the previous section.

3 Applying the design method

3.1 Preliminary remarks

Although the case study concerns a plant of limited complexity, it nevertheless supports a considerable range of functions that are needed for its correct operation. In order to produce batches, it must be possible to fill the plant with the necessary material in an initial step. Its production can either be controlled fully automatically, or be supervised by an operator, who should have only the possibility to guide the plant along "safe" executions. Moreover, it should be possible to shut down the plant, e.g. in the case of failure. Additionally, there is the need for a cleaning procedure, because of potential salt deposits blocking the valves. In general, one also has to deal with fluid losses through leakage, wear, breakage, etc.

In this paper we confine ourselves to the basic functionality of the plant and investigated the central process that given an initial load will produce batches "forever". We show how, for this process, a formal definition can be given, and an implementation can be derived and verified. It should be relatively straightforward to add the other features to our "core" design.

The batch plant of the case study is not intended for real production, and therefore follows a recycling process: the salt solution that is "produced" is in fact the raw material for an inverse procedure that yields the ingredients for a following production. This cyclical operation of the plant introduces some odd effects:

1. In order to specify the production cycle it must be cut at some point, so that we get a production process with an initial state and a final state. The initial state inducess a precondition, and the final state a postcondition. As we require a fully cyclical process, it must always be the case that the precondition implies the postcondition. For such specifications there is always a trivial implementation, i.e. "do nothing". For the case study we will have to keep in mind that this naive solution is not the intended one.

2. It is difficult to give a precise definition of what a batch is. For a single batch this may still be easy: for example, a batch starts with concentrated salt solution in container B1 and water in container B2, and finally there is again concentrated salt solution in container B7 (or B1) and water in B6 (or B2). For multiple batch production this is not obvious any more: the initial condition of the plant is that there are amounts of water and salt solutions of different concentrations distributed over different containers of the plant (in a consistent manner). In this context it is somewhat arbitrary to define what a batch is: in fact, as long as the plant works (and does not deadlock) it "produces batches". In our approach here we define that a batch is produced if container B3 is emptied and filled again.

The reader should keep in mind that these odd effects are due to the academic nature of the plant. We will not pay too much attention to such "artificial" difficulties in the remainder.

We now demonstrate how the specification method can be applied to the case study. Following this procedure we first will specify in section 3.2 the system requirement ("produce batches"). We then develop a specification of the plant that makes this requirement true. The detailed design decisions can be found in section 3.3. The plant specification will be a conjunction of specifications of "basic" production steps (e.g. "transport from B1 to B3"). Subsequently, in section 3.4 a control specification will be derived that forces the plant to behave as stated by the system requirements. The control specification consists of a set of processes, each one responsible for one of the production steps. It will turn out that to guarantee the required behaviour also mutual exclusion mechanism must be added to the control process specifications.

3.2 Specification of the requirement

The most general statement of what the system should do is that it should continuously produce batches. Bearing the remarks of the above section in mind, we can start to write this down in our logical formalism as the following requirement:

$$\text{Production\_of\_Batches} =_{df} \Box \Diamond \text{ Diluted\_salt\_solution\_in\_B3} \tag{4}$$

where Diluted_salt_solution_in_B3 ia an abstract proposition with the obvious meaning, subject to further refinement. In the sequel we will introduce a number of such abstract propositions, whose names indicate their informal meaning. They come with a domain-specific theory of intuitively obvious properties, e.g. a stipulation that containers cannot have different contents at the same time, that the contents of containers does not change spontaneously, etc. The only less intuitive exception to the rule that we need is that we formally allow mixing tank B3 to contain two different liquids prior to mixing. To keep our presentation within acceptable limits we consider this theory as implicitly given.

In principle, (4) allows also for Zeno-like behaviour, i.e. the production of infinitely many batches in finite time. Such behaviour, of course, is unimplementable. Our refined specification of plant and controller will be such, therefore, that a non-Zeno solution is guaranteed to exist. Thus, our current starting point is:

$$\text{Theorem} =_{\text{def}} \quad \text{Spec\_Plant} \Rightarrow \text{ Production\_of\_Batches} \tag{5}$$

In the sequel we will redefine this theorem and the specifications contained in it several times. To make explicit which version of a specification is meant we use the number of the defining equation as its index. Where there is no index, we always mean the most recent version.

3.3 Specification of the plant

There are two complementary approaches to derive a plant specification from the piping and instrumentation diagram of figure 1. One is to come up with generic specifications of a valve, a container, a pipe etc., and then to combine instantiations of these specifications according to the diagram. An advantage of this approach is that it is rather straightforward, and it would not be difficult to have a tool supporting such a specification process. The main disadvantage is that

the resulting specifications are often big and not very structured, at least not in the way we need for the further specification process. It typically includes uninteresting behaviour, such as opening and closing a valve under an empty container. To verify specifications of this kind such superfluous behaviour has to be eliminated to get models of an acceptable size (cf. [6]).

The other specification approach makes use of the designer's knowledge of the plant and its intended behaviour. The result is a structured plant specification that simplifies the design of the control specification, as it contains only information that is needed. Introducing knowledge of the plant in its specification is certainly not a deterministic process: it depends on the view and creativity of the designer. This must be supported by a structured way of specification that makes the assumptions, choices, and consequences explicit and comprehensable. In this way a non-trivial plant specification can be derived through a sequence of clear steps. It is this approach to specification that we wil try to follow.

As a starting point of the design process we choose the insight that the plant does two things (cyclically): it mixes two ingredients and then seperates them again. We add the details listed below to the specification in a stepwise fashion.

- The processes "mixing" and "separation" are split into finer process steps until we reach a level of basic process specifications. In our case they will be transport steps between containers (where evaporation is a special case of a transport step), and cooling steps.
- For each of the basic processes we introduce conditions on the filling levels of the tanks: e.g. transport steps can only take place if there is sufficient solution (or water) in the upper tank and sufficient free space in the lower one. Because we consider batches of fixed size only, and (transportation of) volumes derived from such fixed sizes, the requirements on the filling levels follow immediately.
- In the last step the conditions about valves, mixer, heater and pumps are introduced. A transport step takes place successfully only if the valves between the tanks are open for a certain period. The time requirements are taken from the plant specification [11].

Splitting 1

A production of a batch consists of first separating the ingredients and then mixing them again. This observation gives the first specification of the plant.

$$\text{Spec\_Plant} =_{df} \text{Separation} \wedge \text{Mixing} \tag{6}$$

$$\text{Separation} =_{df} \Box\,(\text{Diluted\_salt\_solution\_in\_B3} \Rightarrow \Diamond\,(\text{Concentrated\_salt\_solution\_in\_B1} \wedge \text{Water\_in\_B2})) \tag{7}$$

$$\text{Mixing} =_{df} \Box\,(\text{Concentrated\_salt\_solution\_in\_B1} \wedge \text{Water\_in\_B2} \Rightarrow \Diamond\,\text{Diluted\_salt\_solution\_in\_B3}) \tag{8}$$

From this plant specification we cannot conclude the production of batches as in (4); we need an initial condition, e.g.

$$\text{Initial\_Condition} =_{df} \text{Diluted\_salt\_solution\_in\_B3} \tag{9}$$

With these definitions our theorem becomes:

$$\text{Theorem} =_{\text{def}} \text{Spec\_Plant}_{(6)} \Rightarrow (\text{Initial\_Condition}_{(9)} \Rightarrow \text{Production\_of\_Batches}_{(4)}) \tag{10}$$

With the definitions above the proof of $\text{Theorem}_{(10)}$ is obvious.

Splitting 2

The next observation that goes into the specification is that separation and mixing consist of several steps:

$$\begin{aligned}
\text{Separation} &=_{df} \text{Transport_to_Evaporator} \\
&\quad \wedge \text{Evaporation} \\
&\quad \wedge \text{Remove_the_Products} \qquad (11)\\
\text{Mixing} &=_{df} \text{Add_concentrated_solution} \\
&\quad \wedge \text{Add_water} \\
&\quad \wedge \text{Mix_ingredients} \qquad (12)\\
&\text{where}\\
\text{Transport_to_Evaporator} &=_{df} \Box\,(\text{Diluted_salt_solution_in_B3} \\
&\qquad \Rightarrow \Diamond\, \text{Diluted_salt_solution_in_B4}) \\
&\quad \wedge \Box\,(\text{Diluted_salt_solution_in_B4} \\
&\qquad \Rightarrow \Diamond\, \text{Diluted_salt_solution_in_B5})\\
\text{Evaporation} &=_{df} \Box\,(\text{Diluted_salt_solution_in_B5} \\
&\qquad \Rightarrow \Diamond\, (\text{Concentrated_salt_solution_in_B5} \wedge\ \text{Water_in_B6}))\\
\text{Remove_the_Products} &=_{df} \Box\,(\text{Concentrated_salt_solution_in_B5} \qquad (13)\\
&\qquad \Rightarrow \Diamond\, \text{Diluted_salt_solution_in_B1}) \\
&\quad \wedge \Box\,(\text{Water_in_B6} \\
&\qquad \Rightarrow \Diamond\, \text{Water_in_B2})\\
&\text{and}\\
\text{Add_concentrated_solution} &=_{df} \Box\,(\text{Concentrated_salt_solution_in_B1} \qquad (14)\\
&\qquad \Rightarrow \Diamond\, \text{Concentrated_salt_solution_in_B3})\\
\text{Add_water} &=_{df} \Box\,(\text{Water_in_B2} \Rightarrow \Diamond\, \text{Water_in_B3})\\
\text{Mix_ingredients} &=_{df} \Box\,((\text{Concentrated_salt_solution_in_B3} \wedge\ \text{Water_in_B3}) \\
&\qquad \Rightarrow \Diamond\, \text{Diluted_salt_solution_in_B3})
\end{aligned}$$

To show that $\text{Theorem}_{(10)}$ holds for the actual version of the plant sepcification there are two possibilities: either we prove that the actual specifications imply the previous ones, or we give a new proof. In the first case we could reuse the proof for $\text{Theorem}_{(10)}$. The second case has to be applied, if due to new insight the actual specification does not imply the previous one.

At this point, we are in the second case: $\text{Separation}_{(11)} \wedge \text{Mixing}_{(12)}$ does not imply $\text{Separation}_{(7)} \wedge \text{Mixing}_{(8)}$. The reason is that, in fact, the specification of $\text{Separation}_{(7)}$ and $\text{Mixing}_{(8)}$ was too strong: it is not necessary that concentrated salt solution and water are in B1, resp. B2, *at*

the same time. This insight emerged naturally in the actual specification of $\text{Separation}_{(11)}$ and $\text{Mixing}_{(12)}$. We redefine $\text{Spec\_plant}_{(11)} =_{df} \text{Separation}_{(11)} \wedge \text{Mixing}_{(12)}$ for which it is easy to show that:

$$\text{Theorem} =_{df} \quad \text{Spec\_plant}_{(11)} \Rightarrow (\text{Initial\_Condition}_{(9)} \Rightarrow \text{Production\_of\_Batches}_{(4)}) \qquad (15)$$

Splitting 3

With respect to the evaporation process we now observe that in the first instance a *hot* concentrated salt solution and *hot* water are produced, which are subsequently cooled down. With this splitting we reach the finest level of production steps: we consider each of the conjuncts of Spec_Plant as a basic process. In subsequent specification steps the basic processes are extended with more detailed information.

In the actual step we get a new version of "Remove_the_products".

$$\begin{aligned}
\text{Remove_the_products} \quad &=_{df} \quad \text{Cool} \wedge \text{Pump} \qquad (16)\\
&\text{where}\\
\text{Cool} \quad &=_{df} \quad \Box\,(\text{Hot_concentrated_salt_solution_in_B5}\\
&\qquad\qquad \Rightarrow \Diamond\, \text{Hot_concentrated_salt_solution_in_B7})\\
&\qquad \wedge \Box\,(\text{Hot_concentrated_salt_solution_in_B7}\\
&\qquad\qquad \Rightarrow \Diamond\, \text{Concentrated_salt_solution_in_B7 })\\
&\qquad \wedge \Box\,(\text{Hot_water_in_B6}\\
&\qquad\qquad \Rightarrow \Diamond\, \text{Water_in_B6})\\
\text{Pump} \quad &=_{df} \quad \Box\,(\text{Concentrated_salt_solution_in_B7}\\
&\qquad\qquad \Rightarrow \Diamond\, \text{Concentrated_salt_solution_in_B1})\\
&\qquad \wedge \Box\,(\text{Water_in_B6}\\
&\qquad\qquad \Rightarrow \Diamond\, \text{Water_in_B2})
\end{aligned}$$

We define $\text{Spec\_plant}_{(16)}$ as $\text{Spec\_plant}_{(11)}$ where $\text{Remove\_the\_products}_{(13)}$ is substituted by $\text{Remove\_the\_products}_{(16)}$. It is easy to see that $\text{Remove\_the\_products}_{(16)}$ implies $\text{Remove\_the\_products}_{(13)}$, and therefore $\text{Spec\_plant}_{(16)}$ implies $\text{Spec\_plant}_{(11)}$. Altogether, the theorem holds:

$$\text{Theorem} =_{df} \quad \text{Spec\_plant}_{(16)} \Rightarrow (\text{Initial\_Condition}_{(9)} \Rightarrow \text{Production\_of\_Batches}_{(4)}) \qquad (17)$$

Conditions about Containers

At this stage we make two decisions:

1. The step from qualitative to quantitative characterisation of the "basic processes" will be taken.
2. By fixing the batch size we restrict the possible behaviour of the plant. This allows for a discrete modelling of this generally continuous aspect of the hybrid behaviour.

We now consider the filling levels of the containers. One can only put something into a container, if there is sufficient space. Some containers have twice the storage capacity of others. The present

design decision is to fix the size of a batch (diluted salt solution) to 7 l. This is the amount that can be stored in container B5. The concentration of the concentrated salt solution is 5 g/l, of the diluted solution 3 g/l. From this it follows that the amount of concentrated salt solution for one batch of 7 l is 4.2 l and the amount of water is 2.8 l. According to these numbers and the capacities of the tanks (see specification of the plant) we can see that containers B1, B2, B4, B7 and B6 have the capacity of two batches, or ingredients for two batches, respectively, and B3 and B5 can contain at most one batch. In the following level of specification we take the different amounts and tank capacities into consideration. In the actual specification we also restrict the behaviour of the plant, i.e. the desired behaviour of the plant that we consider. The restriction is that we exclude some of the potential parallel behaviour; we exclude the possibility that water and concentrated solution are filled to B3 in parallel, or other tanks are filled and emptied at the same time.

We write B3 = Sol42C to indicate that the content of container B3 is 4.2 l concentrated, cold salt solution; B3 = sol70C indicates that there are 7 l diluted cold salt solution in container B3, etc..

To abbreviate the specification of symmetric cases we mention alternative values between brackets. These steps have two interpretations: for each container if there is an alternative then either read all the unbracketed values or read all the values in the brackets throughout.

In general, one could continue to reason about the subsystems identified (e.g. Mixing, or at another level of detail Remove_the_Products), and their interconnections. For large systems this modular perspective is a prerequisite to cope with complexity. In this case we can afford to consider the the "flattened" specification of Spec_Plant:

$$
\begin{aligned}
&\text{Spec\_Plant} =_{df} && (18)\\
&\quad \Box\ (\ \text{B3} = \text{sol70C} \wedge \text{B4} = \text{empty (sol70C)} && (\text{S1})\\
&\qquad \Rightarrow\ \Diamond\ \text{B3} = \text{empty and B4} = \text{sol70C (sol140C)}\ )\\
&\quad \wedge \Box\ (\ \text{B4} = \text{sol70C (sol140C)} \wedge \text{B5} = \text{empty} && (\text{S2})\\
&\qquad \Rightarrow\ \Diamond\ \text{B4} = \text{empty (sol70C)} \wedge \text{B5} = \text{sol70C})\\
&\quad \wedge \Box\ (\ \text{B5} = \text{sol70C} \wedge \text{B6} = \text{empty (water28C} \vee \text{water28H)} && (\text{S3})\\
&\qquad \Rightarrow\ \Diamond\ \text{B5} = \text{Sol42H} \wedge \text{B6} = \text{water28H (water56H))}\\
&\quad \wedge \Box\ (\ \text{B5} = \text{Sol42H} \wedge \text{B7} = \text{empty (Sol42C} \vee \text{Sol42H)} && (\text{S4})\\
&\qquad \Rightarrow\ \Diamond\ \text{B5} = \text{empty} \wedge \text{B7} = \text{Sol42H (Sol84H)}\ )\\
&\quad \wedge \Box\ (\ \text{B7} = \text{Sol42H (Sol84H)} && (\text{S5})\\
&\qquad \Rightarrow\ \Diamond\ \text{B7} = \text{Sol42C (Sol84C)}\ )\\
&\quad \wedge \Box\ (\ \text{B6} = \text{water28H (water56H)} && (\text{S6})\\
&\qquad \Rightarrow\ \Diamond\ \text{B6} = \text{water28C (water56C)}\ )\\
&\quad \wedge \Box\ (\ \text{B7} = \text{Sol42C (Sol84C)} \wedge \text{B1} = \text{empty (Sol42C)} && (\text{S7})\\
&\qquad \Rightarrow\ \Diamond\ \text{B7} = \text{empty (Sol42C)} \wedge \text{B1} = \text{Sol42C (Sol84C)}\ )\\
&\quad \wedge \Box\ (\ \text{B6} = \text{water28C (water56C)} \wedge \text{B2} = \text{empty (water28C)} && (\text{S8})\\
&\qquad \Rightarrow\ \Diamond\ \text{B6} = \text{empty (water28C)} \wedge \text{B2} = \text{water28C (water56C)}\ )\\
&\quad \wedge \Box\ (\ \text{B1} = \text{Sol42C (Sol82C)} \wedge \text{B3} = \text{empty} && (\text{S9})\\
&\qquad \Rightarrow\ \Diamond\ \text{B1} = \text{empty (Sol42C)} \wedge \text{B3} = \text{Sol42C)}\\
&\quad \wedge \Box\ (\ \text{B2} = \text{water28C (water56C)} \wedge \text{B3} = \text{empty} && (\text{S10})\\
&\qquad \Rightarrow\ \Diamond\ \text{B2} = \text{empty (water28)} \wedge \text{B3} = \text{water28C)}\\
&\quad \wedge \Box\ (\ \text{B1} = \text{Sol42C (Sol82C)} \wedge \text{B3} = \text{water28C)} && (\text{S11})\\
&\qquad \Rightarrow\ \Diamond\ \text{B1} = \text{empty (Sol42C)} \wedge \text{B3} = \text{sol70C)}\\
&\quad \wedge \Box\ (\ \text{B2} = \text{water28C (water56C)} \wedge \text{B3} = \text{Sol42C} && (\text{S12})\\
&\qquad \Rightarrow\ \Diamond\ \text{B2} = \text{empty (water28C)} \wedge \text{B3} = \text{sol70C)}
\end{aligned}
$$

At this point we make the following two observations:

1. We have to adjust the initial condition: currently neither $\text{Spec\_plant}_{(18)} \Rightarrow \text{Spec\_plant}_{(16)}$ holds, nor does $\text{Spec\_plant}_{(18)} \Rightarrow (\text{Initial\_Condition}_{(9)} \Rightarrow \text{Production\_of\_Batches}_{(4)})$.
2. We now have the knowledge to strengthen the original requirement $\text{Production\_of\_Batches}_{(4)}$ and exclude the trivial solution that there is a batch in B3 and never something happens.

$\text{Initial\_Condition}_{(9)}$ is stronger than necessary: it requires that "one batch" is present in container B3. However, it could also be another container. In order to produce batches, there must be at least water and salt solution for "one batch" corresponding to 7 l of diluted salt solution somewhere in the plant. Additionally, it is easy to see that there must be one batch worth of free space in the plant to keep the system moving, i.e. such that at least one of the steps S1, ..., S12 can be performed. Of course, the initial condition for an optimal throughput is certainly more restricted; this initial condition just suffices to show that the system does not deadlock and that batches are produced.

We redefine the initial condition as:

$$
\begin{array}{lllll}
\text{Initial\_Condition} =_{df} & & & & (19) \\
 & (& (& (\neg\ \text{B1=empty} \vee \neg\ \text{B7=empty} \vee \text{B3=Sol42C} \vee \text{B5=Sol42H}) & \\
 & & \wedge & (\neg\ \text{B2=empty} \vee \neg\ \text{B6=empty} \vee \text{B3=water28C}) & \\
 & &) & & \\
 & \vee & & (\neg\ \text{B3=sol70C} \vee \neg\ \text{B4=empty} \vee \text{B5=sol70C}) & \\
 &) & & & \\
\wedge & (& (& (\neg\ \text{B1=Sol84C} \vee \neg\ (\text{B7=Sol84C} \vee \text{B7=Sol84H}) \vee \text{B3=water28C}) & \\
 & & \wedge & (\neg\ \text{B2=water56C} \vee \neg\ (\text{B6=water56C} \vee \text{B6=water56H}) \vee \text{B3=Sol42C}) & \\
 & &) & & \\
 & \vee & & (\text{B3=empty} \vee \neg\ \text{B4=sol140C} \vee \text{B5=empty}) & \\
 &) & & &
\end{array}
$$

Unfortunately, this predicate describes about 8100 possible different initial states. In fact, we will not be able to deal quite with the initial condition as it is stated here, and follow an alternative approach described later. For the time being, however, we maintain the above definition as the ideal specification of the allowed initial states of the plant.

To exclude trivial solutions we have to strengthen the requirement. When we require that B3 has to be filled infinitely often with a batch, we assume implicitly that infinitely often a batch is removed from B3. This can be made explicit:

$$\text{Production\_of\_Batches} =_{df} \Box \Diamond\ \text{Diluted\_salt\_solution\_in\_B3} \wedge \Box \Diamond\ \text{B3\_is\_empty} \qquad (20)$$

Our conjecture is that the new version of the theorem is provable.

$$\text{Theorem} =_{df} \quad \text{Spec\_plant}_{(18)} \Rightarrow (\text{Initial\_Condition}_{(19)} \Rightarrow \text{Production\_of\_Batches}_{(20)}) \qquad (21)$$

Conditions about Valves, Heater, Mixer and Pumps

We have now arrived at a refinement of the specification of the plant into twelve production steps S1, ..., S12 which are so "basic" that one can attempt to validate whether the corresponding parts of the physical plant really implement them. Looking at figure 1 we see that this is not the case. To make the specifications complete we will have to consider the valves, pumps, heater and mixer. A transport step from one tank to another, as specified above, can only take place if the valve between the containers is open until the filling level in the second tank is reached (the postcondition holds). The evaporation step requires the heater to be on. As a consequence of the previous design decision excluding some of the potential behaviour, we have to assume restricted behaviour of the valves: if a tank is filled it may not be emptied at the same time. This restriction is fulfilled, if we never allow for a tank that the valve above and below are open at the same time.

The previous specification for each of the steps was of the form:

$$\Box(\text{precondition} \Rightarrow \Diamond\ \text{postcondition}) \tag{22}$$

We now arrive at form (3) as explained in section 2.2.

$$\text{Spec\_Plant} =_{df} \tag{23}$$

$$\Box(\ \text{B3} = \text{sol70C} \wedge \text{B4} = \text{empty (sol70C)} \tag{S1}$$
$$\Rightarrow (\Box\ \ (\text{V11} = \text{open} \wedge \text{V8} = \text{closed} \wedge \text{V9} = \text{closed} \wedge \text{V12} = \text{closed})$$
$$\Rightarrow \Diamond\ (\ \text{B3} = \text{empty and B4} = \text{sol70C (sol140C)}\ )\ )$$

$$\wedge\quad \Box(\ \text{B4} = \text{sol70C (sol140C)} \wedge \text{B5} = \text{empty} \tag{S2}$$
$$\Rightarrow (\Box\ \ (\text{V12} = \text{open} \wedge \text{V11} = \text{closed} \wedge \text{V15} = \text{closed} \wedge \text{heater} = \text{off}\ )$$
$$\Rightarrow \Diamond\ (\ \text{B4} = \text{empty (sol70C)} \wedge \text{B5} = \text{sol70C})\ )$$

$$\wedge\quad \Box(\ \text{B5} = \text{sol70C} \wedge \text{B6} = \text{empty (water28C, water28H)} \tag{S3}$$
$$\Rightarrow (\Box\ \ (\ \text{heater} = \text{on} \wedge \text{V13} = \text{open} \wedge \text{V12} = \text{closed} \wedge \text{V15} = \text{closed}$$
$$\wedge\ \text{V20} = \text{closed})$$
$$\Rightarrow \Diamond\ (\ \text{B5} = \text{Sol42H} \wedge \text{B6} = \text{water28H (water56H)})$$

$$\wedge\quad \Box(\ \text{B5} = \text{Sol42H} \wedge \text{B7} = \text{empty (Sol42C,Sol42H)} \tag{S4}$$
$$\Rightarrow (\Box\ \ (\text{V15} = \text{open} \wedge \text{V12} = \text{closed} \wedge \text{V18} = \text{closed} \wedge \text{heater} = \text{off})$$
$$\Rightarrow \Diamond\ (\ \text{B5} = \text{empty} \wedge \text{B7} = \text{Sol42H (Sol84H)}\ )\ )$$

$$\wedge\quad \Box(\ \text{B7} = \text{Sol42H (Sol84H)} \tag{S5}$$
$$\Rightarrow (\Box\ \ (\text{V17} = \text{open} \wedge \text{V15} = \text{closed} \wedge \text{V18} = \text{closed})$$
$$\Rightarrow \Diamond\ (\ \text{B7} = \text{Sol42C (Sol84C)}\ )\ )$$

$$\wedge\quad \Box(\ \text{B6} = \text{water28H (water56H)} \tag{S6}$$
$$\Rightarrow (\Box\ \ (\text{V29} = \text{open} \wedge \text{V13} = \text{closed} \wedge \text{heater} = \text{off} \wedge \text{V20} = \text{closed})$$
$$\Rightarrow \Diamond\ (\ \text{B6} = \text{water28C (water56C)}\ )\ )$$

$$\wedge\quad \Box(\ \text{B7} = \text{Sol42C (Sol84C)} \wedge \text{B1} = \text{empty (Sol42C)} \tag{S7}$$
$$\Rightarrow (\Box\ \ (\text{V18} = \text{open} \wedge \text{V23} = \text{open} \wedge \text{V22} = \text{open} \wedge \text{V1} = \text{open} \wedge \text{V3} = \text{open}$$
$$\wedge\ \text{P1} = \text{on} \wedge \text{V15} = \text{closed} \wedge \text{V17} = \text{closed} \wedge \text{V8} = \text{closed})$$
$$\Rightarrow \Diamond\ (\ \text{B7} = \text{empty (Sol42C)} \wedge \text{B1} = \text{Sol42C (Sol84C)}\ )\ )$$

$$\wedge\quad \Box(\ \text{B6} = \text{water28C (water56C)} \wedge \text{B2} = \text{empty (water28C)} \tag{S8}$$
$$\Rightarrow (\Box\ \ (\text{V20} = \text{open} \wedge \text{V24} = \text{open} \wedge \text{V25} = \text{open} \wedge \text{V5} = \text{open} \wedge \text{V6} = \text{open}$$
$$\text{und P2} = \text{on} \wedge \text{V13} = \text{closed} \wedge \text{heater} = \text{off} \wedge \text{V9} = \text{closed})$$

$$\Rightarrow \Diamond \ (\ \mathrm{B6} = \mathrm{empty\ (water28C)} \wedge \mathrm{B2} = \mathrm{water28C\ (water56C)}\ )\ )$$

$$\Box(\ \mathrm{B1} = \mathrm{Sol42C\ (Sol82C)} \wedge \mathrm{B3} = \mathrm{empty} \qquad \text{(S9)}$$
$$\Rightarrow (\Box \quad (\mathrm{V8} = \mathrm{open} \wedge \mathrm{V3} = \mathrm{closed} \wedge \mathrm{V9} = \mathrm{closed} \wedge \mathrm{V11} = \mathrm{closed})$$
$$\Rightarrow \Diamond \ (\ \mathrm{B1} = \mathrm{empty\ (Sol42C)} \wedge \mathrm{B3} = \mathrm{Sol42C})\ )$$

$$\wedge \quad \Box(\ \mathrm{B2} = \mathrm{water28C\ (water56C)} \wedge \mathrm{B3} = \mathrm{empty} \qquad \text{(S10)}$$
$$\Rightarrow (\Box \quad (\mathrm{V9} = \mathrm{open} \wedge \mathrm{V6} = \mathrm{closed} \wedge \mathrm{V8} = \mathrm{closed} \wedge \mathrm{V11} = \mathrm{closed})$$
$$\Rightarrow \Diamond \ (\ \mathrm{B2} = \mathrm{empty\ (water28)} \wedge \mathrm{B3} = \mathrm{water28C})\ )$$

$$\wedge \quad \Box(\ \mathrm{B1} = \mathrm{Sol42C\ (Sol82C)} \wedge \mathrm{B3} = \mathrm{water28C}) \qquad \text{(S11)}$$
$$\Rightarrow (\Box \quad (\mathrm{V8} = \mathrm{open} \wedge \mathrm{Mixer} = \mathrm{on} \wedge \mathrm{V3} = \mathrm{closed} \wedge \mathrm{V9} = \mathrm{closed} \wedge \mathrm{V11} = \mathrm{closed})$$
$$\Rightarrow \Diamond \ (\ \mathrm{B1} = \mathrm{empty\ (Sol42C)} \wedge \mathrm{B3} = \mathrm{sol70C})\ )$$

$$\wedge \quad \Box(\ \mathrm{B2} = \mathrm{water28C\ (water56C)} \wedge \mathrm{B3} = \mathrm{Sol42C} \qquad \text{(S12)}$$
$$\Rightarrow (\Box \quad (\mathrm{V9} = \mathrm{open} \wedge \mathrm{Mixer} = \mathrm{on} \wedge \mathrm{V6} = \mathrm{closed} \wedge \mathrm{V8} = \mathrm{closed} \wedge \mathrm{V11} = \mathrm{closed})$$
$$\Rightarrow \Diamond \ (\ \mathrm{B2} = \mathrm{empty\ (water28C)} \wedge \mathrm{B3} = \mathrm{sol70C})\ )$$

A result of our approach to the specification of the basic production steps S1,...,S12 is that they do not require valves being opened or closed during their execution; they only state what is required to happen while some of them are open or closed during the entire step. This format makes it particularly simple to validate our specification against the physical plant, where twelve simple experiments suffice: adjust the valves as specified in the invariants, realise the precondition, and check whether after some time the postcondition becomes true. This is important because they suffice to prove the desired production of batches, which means that we only need to model such simple properties to design our controller. For our correctness requirement we do not need to consider more complicated potential behaviour of the plant.

In practice, the basic production steps satisfy even stronger properties: they have *bounded liveness* properties, i.e. if precondition and invariant are fulfilled then the postcondition will hold after a time duration with a fixed upper bound. The plant description in [11] in fact gives the durations of all basic processes. A possible next refinement of our plant specification could be to include such upper bounds, either explicitly, following the table, or more abstractly as uninstantiated constants. We will not do so here, as it is not needed to prove our correctness criterion. We will return to this point later, however, as the time-bounded production of batches is a practically interesting property, of course.

Our final plant specification above does not yet imply the previous specification $\mathrm{Spec\_Plant}_{(18)}$, nor does it make $\mathrm{Theorem}_{(21)}$ hold. The reason is that the twelve production steps as specified only "work" as long as the environment ensures that the corresponding invariants hold. The control program that we are going to design must take care of this.

3.4 Specification of the control program

In $\mathrm{Theorem}_{(21)}$ we want to substitute $\mathrm{Spec\_Plant}_{(18)}$ by $\mathrm{Spec\_Plant}_{(23)}$, which is considerably weaker: its conjuncts are of form (3) rather than (22). To compensate we can add for each of the conjuncts of $\mathrm{Spec\_Plant}_{(23)}$ a "control fragment"

$$\Box\ (\ \mathrm{precondition} \Rightarrow\ \mathrm{invariant}\ \mathit{Until}\ \mathrm{result}) \qquad (24)$$

to the left side of the theorem.

The conjunction of these control fragments is our first approximation of a specification of a control program, consisting of twelve parallel processes. This specification is unimplementable, however,

as it requires certain valves to be open and closed at the same time. We have to provide a mutual exclusion mechanism.

Introducing mutual exclusion

To satisfy an invariant, all the resources that contribute to the invariant, such as the valves, pumps, mixer and heater, have to be treated as critical objects that may be accessed by only one of the control processes at a time. We assume that for two processes P_i and P_j that want to access the same resources the predicates "claimed_resources_i" and "claimed_resources_j" never can be true at the same time for different i and j.

A better approximation of the control program specification is therefore of the form:

$$\Box\ (\text{precondition\_}i \wedge\ \text{claimed\_resources\_}i \\ \Rightarrow\ ((\text{invariant\_}i \wedge\ \text{claimed\_resources\_}i)\ \mathit{Until}\ \ \text{result\_}i)) \tag{25}$$

To make sure that resources are released when a process is ready, we add the extra requirement:

$$\Box\ (\text{result\_}i \Rightarrow\ \neg\text{claimed\_resources\_}i) \tag{26}$$

Towards the programmable logic controller

In the last design step we take the special technology into account on which the control program will be executed: a Programmable Logic Controller with the programming language Sequential Function Charts [10, 18]. This language allows to express that processes are excuted in parallel, and for each process there is a system defined variable that indicates whether the process is *active* or not.

This last design step includes also a reformulation of the previous specification and a change in the point of view: from resources that may only be accessed by one process to that of *conflicting* processes. We regard a process as being active if its resources have been claimed and its invariant holds. In that case the corresponding production step really "produces", e.g. the solution flows in fact from one container into another. The condition for the activation of a process is that the precondition of (25) is satisfied, and that the required resources can be claimed. We reformulate the format of (25) (and (26)) as we want to allow for stronger preconditions and invariants that imply the claiming of resources.

$$\Box\ (\text{precondition}'\text{\_}i \Rightarrow\ \text{invariant}'\text{\_}i\ \mathit{Until}\ \ \text{result\_}i) \tag{27}$$
$$\wedge\quad \Box\ (\text{precondition}'\text{\_}i \Rightarrow\ \text{claimed\_resources\_}i) \tag{28}$$
$$\wedge\quad \Box\ (\text{invariant}'\text{\_}i \Rightarrow\ \text{claimed\_resources\_}i) \tag{29}$$
$$\wedge\quad \Box\ (\text{result\_}i \Rightarrow\ \neg\text{claimed\_resources\_}i) \tag{30}$$
$$\wedge\quad \Box\ (\text{precondition}'\text{\_}i \Rightarrow\ \neg\text{result\_}i) \tag{31}$$

The last condition (31) is meant to prevent the preconditions from becoming too strong, so that the result would already hold and we would obtain a static solution.

In the rest of this section we work out the predicates precondition$'$_i and claimed_resources$'$_i. To do so, we change perspective from critical resources to conflicting processes. We regard two processes as conflicting if their invariants are inconsistent. Instead of the requirement that each resource

may only be claimed by one process at a time, we equivalently require that of each conflicting pair of processes at most one may be active. The solution to the mutual exclusion problem is such that the three conditions have to be fulfilled when a process is activated, together resulting in the precondition′ above:

1. the container filling conditions for a process are satisfied (the preconditions above); and
2. no conflicting process is currently active; and
3. of all non-active conflicting processes that satisfy 1) and 2) only one may be activated.

We solve 3) in a very simply way by giving priorities to conflicting pairs of processes. Table 1 lists the possibly conflicting processes, where control process P_i corresponds to production step S_i. The priority graph in figure 2 shows that there are no cycles that could block conflicting processes and cause deadlocks.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
P1		p							#	#	#	#
P2	#		p	p								
P3		#		#		#		#				
P4		#	p		#		#					
P5				p			#					
P6			p					#				
P7				p	p				p		p	
P8			p			p				p		p
P9	p						#			#		#
P10	p							#	p		p	
P11	p						#			#		#
P12	p							#	p		p	

Table 1: Conflicting steps and priorities. # and p indicate a conflict, where p at (Pi,Pj) means that Pi has priority over Pj.

Filling conditions

For a process to be activated the filling conditions of the tanks must be satisfied. For process P_i the predicate Φ_i below expresses the preconditions as in Spec_Plant(23). To reduce their representation

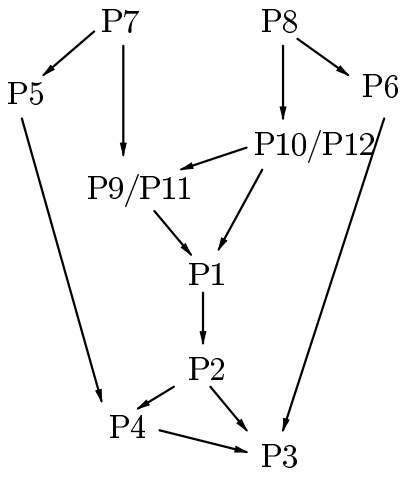

Figure 2: Priority graph derived from Table 1. An arrow from Pi to Pj is drawn when Pj has priority over Pi. Transitive arrows are left out.

we use the same notational abbreviation as above and write alternative values for symmetric cases between brackets. Therefore, predicates with brackets must be interpreted as predicate schemes that represent 2 or 4 different predicates.

$\Phi_1 =_{df}$ B3 = sol70C $\wedge$ B4 = empty (sol70C)
$\Phi_2 =_{df}$ B4 = sol70C (sol140C) $\wedge$ B5 = empty
$\Phi_3 =_{df}$ B5 = sol70C $\wedge$ B6 = empty (water28C $\vee$ water28H)
$\Phi_4 =_{df}$ B5 = Sol42H $\wedge$ B7 = empty (Sol42C $\vee$ Sol42H)
$\Phi_5 =_{df}$ B7 = Sol42H $\vee$ B7 = Sol84H
$\Phi_6 =_{df}$ B6 = water28H $\vee$ B6 = water56H
$\Phi_7 =_{df}$ B7 = Sol42C (Sol84C) $\wedge$ B1 = empty (Sol42C)
$\Phi_8 =_{df}$ B6 = water28C (water56C) $\wedge$ B2 = empty (water28C)
$\Phi_9 =_{df}$ B1 = Sol42C (Sol82C) $\wedge$ B3 = empty
$\Phi_{10} =_{df}$ B2 = water28C (water56C) $\wedge$ B3 = empty
$\Phi_{11} =_{df}$ B1 = Sol42C (Sol82C) $\wedge$ B3 = water28C)
$\Phi_{12} =_{df}$ B2 = water28C (water56C) $\wedge$ B3 = Sol42C

Activity of conflicting processes

A process must not be actived if conflicting processes are already active. The condition Ψ_i thus express that the filling conditions of process Pi are fulfilled *and* no conflicting processes are already active. SFC provides a variable Pi.X for each process Pi which is true if and only if Pi is active. This Pi.X corresponds to claimed_resources_i as above.

The expressions below are constructed using table 1.

$\Psi_1 =_{df} \Phi_1 \wedge \neg$ P2.X $\wedge \neg$ P9.X $\wedge \neg$ P10.X $\wedge \neg$ P11.X $\wedge \neg$ P12.X
$\Psi_2 =_{df} \Phi_2 \wedge \neg$ P1.X $\wedge \neg$ P3.X $\wedge \neg$ P4.X
$\Psi_3 =_{df} \Phi_3 \wedge \neg$ P2.X $\wedge \neg$ P4.X $\wedge \neg$ P6.X $\wedge \neg$ P8.X
$\Psi_4 =_{df} \Phi_4 \wedge \neg$ P2.X $\wedge \neg$ P3.X $\wedge \neg$ P5.X $\wedge \neg$ P7.X
$\Psi_5 =_{df} \Phi_5 \wedge \neg$ P4.X $\wedge \neg$ P7.X

$$
\begin{aligned}
\Psi_6 &=_{df} \Phi_6 \wedge \neg \text{P3.X} \wedge \neg \text{P8.X} \\
\Psi_7 &=_{df} \Phi_7 \wedge \neg \text{P4.X} \wedge \neg \text{P5.X} \wedge \neg \text{P9.X} \wedge \neg \text{P11.X} \\
\Psi_8 &=_{df} \Phi_8 \wedge \neg \text{P3.X} \wedge \neg \text{P6.X} \wedge \neg \text{P10.X} \wedge \neg \text{P12.X} \\
\Psi_9 &=_{df} \Phi_9 \wedge \neg \text{P1.X} \wedge \neg \text{P7.X} \wedge \neg \text{P10.X} \wedge \neg \text{P12.X} \\
\Psi_{10} &=_{df} \Phi_{10} \wedge \neg \text{P1.X} \wedge \neg \text{P8.X} \wedge \neg \text{P9.X} \wedge \neg \text{P11.X} \\
\Psi_{11} &=_{df} \Phi_{11} \wedge \neg \text{P1.X} \wedge \neg \text{P7.X} \wedge \neg \text{P10.X} \wedge \neg \text{P12.X} \\
\Psi_{12} &=_{df} \Phi_{12} \wedge \neg \text{P1.X} \wedge \neg \text{P8.X} \wedge \neg \text{P9.X} \wedge \neg \text{P11.X}
\end{aligned}
$$

Conflicting processes that can be activated

Process Pi may be activated if predicate Ψ_i holds and no conflicting process that has a higher priority can be activated. The expressions Θ_i below are also derived using table 1. They represent the predicates precondition$'_i$ satisfying (27), (28) and (31).

$$
\begin{aligned}
\Theta_1 &=_{df} \Psi_1 \wedge \neg \Psi_2 \\
\Theta_2 &=_{df} \Psi_2 \wedge \neg \Psi_3 \wedge \neg \Psi_4 \\
\Theta_3 &=_{df} \Psi_3 \\
\Theta_4 &=_{df} \Psi_4 \wedge \neg \Psi_3 \\
\Theta_5 &=_{df} \Psi_5 \wedge \neg \Psi_4 \\
\Theta_6 &=_{df} \Psi_6 \wedge \neg \Psi_3 \\
\Theta_7 &=_{df} \Psi_7 \wedge \neg \Psi_4 \wedge \neg \Psi_5 \wedge \neg \Psi_9 \wedge \neg \Psi_{11} \\
\Theta_8 &=_{df} \Psi_8 \wedge \neg \Psi_3 \wedge \neg \Psi_6 \wedge \Psi_{10} \wedge \neg \Psi_{12} \\
\Theta_9 &=_{df} \Psi_9 \wedge \neg \Psi_1 \\
\Theta_{10} &=_{df} \Psi_{10} \wedge \neg \Psi_1 \wedge \neg \Psi_9 \wedge \neg \Psi_{11} \\
\Theta_{11} &=_{df} \Psi_{11} \wedge \neg \Psi_1 \\
\Theta_{12} &=_{df} \Psi_{12} \wedge \neg \Psi_1 \wedge \neg \Psi_9 \wedge \neg \Psi_{11}
\end{aligned}
$$

Now, we have constructed predicates that should make it possible to fulfil the control specifications (27)-(31). In detail, we assign the following expressions for $1 \leq i \leq 12$:

$$
\begin{aligned}
\text{claimed\_resources\_}i &=_{df} \text{P}i\text{.X} && (32) \\
\text{precondition}'\text{\_}i &=_{df} \Theta_i && (33) \\
\text{invariant}'\text{\_}i &=_{df} \text{invariant\_}i \wedge \text{P}i\text{.X} && (34) \\
\text{invariant\_}i &=_{df} \Phi_i && (35)
\end{aligned}
$$

The last observation is, that the control specification so far is still not sufficient. There is some liveness-condition missing: in the specification so far, the predicate claimed_resources_i never needs to become true. According to the considerations above we add the requirements: if precondition$'_i$ holds for process Pi, then process Pi eventually should get active (i.e. Pi.X becomes true).

$$
\text{Liveness} =_{df} \bigwedge_{i=1}^{12} \Box \, (\Box \, \Theta_i \Rightarrow \Diamond \, \text{P}i\text{.X}) \qquad (36)
$$

3.5 The definitive design theorem

We are now able to state the definitive version of the specification theorem. It can be found in figures 3 and 4. The former contains the assumptions of Spec_Plant$_{(23)}$, the latter those of Spec_Control and the right-hand side of the implication, with:

$$\text{Production\_of\_Batches} =_{df} \Box \Diamond \text{B3} = \text{Sol70C} \wedge \Box \Diamond \text{B3} = \text{empty} \tag{37}$$

Note that corresponding fragments of both specifications (i.e. production steps vs. control processes) have been given in the same top-down order. The predicates Θ_i in table 4 are predicates schemes and for each container consistently either the unbracketed or the bracketed value has to be read throughout the table.

It should be observed that we have specified a controller that is independent of the actual durations of the basic plant processes. The use of the *Until*-operator makes clear that the controller would only have Zeno behaviour if the plant does. The latter is not the case, of course.

Having used the used the development of the design theorem to derive a specification of the control process, we now turn to development of the control program itself. After that we will discuss the verification of the design.

4 The implementation of the control program

Before explaining the actual implementation of the control program, we briefly introduce PLC languages and the PLC execution mechanism. For a detailed introduction see, e.g. [18].

4.1 PLC languages and execution mechanism

Sequential Function Charts (SFC) is one of the languages from the standard IEC 61131-3 [10], describing a set of programming languages for PLCs. In fact, SFC is a graphical language that allows to impose structure upon any of the other languages: a Petri-net like execution scheme, where in a sort of token game program steps are activated and deactivated. It is especially useful for our approach, because it allows a form of parallelism.

Any SFC-program consists of a set of *steps* and *transitions* between steps. The program starts in an *initial step*, which is active in the first execution of the program. If the *transition condition* of a step holds, the activity moves to its successor step. There is also the possibility of branching: In an *alternative branch* activity moves to one of the successors, in a *parallel branch* all successors get active. In our control program (see e.g. figure 5) there is one parallel branch.

To each step a program is attached, which in the case here is written in Instruction Lists (IL), an assembly-like language. As long as a step is active the program that belongs to this step is executed. More than one step be active at a time.

In a PLC a program is executed in a permanent loop called the *scan cycle*. In the beginning of each scan cycle the PLC updates its input and output data: it reads the signals at its input ports and writes data to the output ports. Then the program in the PLC is executed once. In the case of SFC programs this means that the set of active steps is determined and each active step's program is executed. Afterwards, a new scan cycle starts.

The typical duration of one scan cycle is a few milliseconds, depending on the size of the program. This scan cycle time determines the "reaction-time" of the PLC control on events in its environment. For some applications reaction time in the range of milliseconds may be crucial. With

	$\Box$	(B3 = sol70C $\wedge$ B4 = empty (sol70C) $\Rightarrow$ ($\Box$ (V11 = open $\wedge$ V8 = closed $\wedge$ V9 = closed $\wedge$ V12 = closed) $\Rightarrow$ $\Diamond$ (B3 = empty and B4 = sol70C (sol140C)))
$\wedge$	$\Box$	(B4 = sol70C (sol140C) $\wedge$ B5 = empty $\Rightarrow$ ($\Box$ (V12 = open $\wedge$ V11 = closed $\wedge$ V15 = closed $\wedge$ heater = off) $\Rightarrow$ $\Diamond$ (B4 = empty (sol70C) $\wedge$ B5 = sol70C))
$\wedge$	$\Box$	(B5 = sol70C $\wedge$ B6 = empty (water28C, water28H) $\Rightarrow$ ($\Box$ (heater = on $\wedge$ V13 = open $\wedge$ V12 = closed $\wedge$ V15 = closed $\wedge$ V20 = closed) $\Rightarrow$ $\Diamond$ (B5 = Sol42H $\wedge$ B6 = water28H (water56H)))
$\wedge$	$\Box$	(B5 = Sol42H $\wedge$ B7 = empty (Sol42C,Sol42H) $\Rightarrow$ ($\Box$ (V15 = open $\wedge$ V12 = closed $\wedge$ V18 = closed $\wedge$ heater = off) $\Rightarrow$ $\Diamond$ (B5 = empty $\wedge$ B7 = Sol42H (Sol84H)))
$\wedge$	$\Box$	(B7 = Sol42H (Sol84H) $\Rightarrow$ ($\Box$ (V17 = open $\wedge$ V15 = closed $\wedge$ V18 = closed) $\Rightarrow$ $\Diamond$ (B7 = Sol42C (Sol84C)))
$\wedge$	$\Box$	(B6 = water28H (water56H) $\Rightarrow$ ($\Box$ (V29 = open $\wedge$ V13 = closed $\wedge$ heater = off $\wedge$ V20 = closed) $\Rightarrow$ $\Diamond$ (B6 = water28C (water56C)))
$\wedge$	$\Box$	(B7 = Sol42C (Sol84C) $\wedge$ B1 = empty (Sol42C) $\Rightarrow$ ($\Box$ (V18 = open $\wedge$ V23 = open $\wedge$ V22 = open $\wedge$ V1 = open $\wedge$ V3 = open $\wedge$ P1 = on $\wedge$ V15 = closed $\wedge$ V17 = closed $\wedge$ V8 = closed) $\Rightarrow$ $\Diamond$ (B7 = empty (Sol42C) $\wedge$ B1 = Sol42C (Sol84C)))
$\wedge$	$\Box$	(B6 = water28C (water56C) $\wedge$ B2 = empty (water28C) $\Rightarrow$ ($\Box$ (V20 = open $\wedge$ V24 = open $\wedge$ V25 = open $\wedge$ V5 = open $\wedge$ V6 = open $\wedge$ P2 = on $\wedge$ V13 = closed $\wedge$ heater = off $\wedge$ V9 = closed) $\Rightarrow$ $\Diamond$ (B6 = empty (water28C) $\wedge$ B2 = water28C (water56C)))
$\wedge$	$\Box$	(B1 = Sol42C (Sol82C) $\wedge$ B3 = empty $\Rightarrow$ ($\Box$ (V8 = open $\wedge$ V3 = closed $\wedge$ V9 = closed $\wedge$ V11 = closed) $\Rightarrow$ $\Diamond$ (B1 = empty (Sol42C) $\wedge$ B3 = S ol42C))
$\wedge$	$\Box$	(B2 = water28C (water56C) $\wedge$ B3 = empty $\Rightarrow$ ($\Box$ (V9 = open $\wedge$ V6 = closed $\wedge$ V8 = closed $\wedge$ V11 = closed) $\Rightarrow$ $\Diamond$ (B2 = empty (water28) $\wedge$ B3 = w ater28C))
$\wedge$	$\Box$	(B1 = Sol42C (Sol82C) $\wedge$ B3 = water28C) $\Rightarrow$ ($\Box$ (V8 = open $\wedge$ Mixer = on $\wedge$ V3 = closed $\wedge$ V9 = closed $\wedge$ V11 = closed) $\Rightarrow$ $\Diamond$ (B1 = empty (Sol42C) $\wedge$ B3 = so l70C))
$\wedge$	$\Box$	(B2 = water28C (water56C) $\wedge$ B3 = Sol42C $\Rightarrow$ ($\Box$ (V9 = open $\wedge$ Mixer = on $\wedge$ V6 = closed $\wedge$ V8 = closed $\wedge$ V11 = closed) $\Rightarrow$ $\Diamond$ (B2 = empty (water28C) $\wedge$ B3 = s ol70C))

Figure 3: Part 1 of the definitive theorem: the plant specification

$\wedge\,\Box$	($\Theta_1 \wedge$ P1.X $\Rightarrow$	((V11 = open $\wedge$ V8 = closed $\wedge$ V9 = closed $\wedge$ V12 = closed) *Until* (B3 = empty and B4 = sol70C (sol140C))))
$\wedge\,\Box$	((B3 = empty and B4 = sol70C (sol140C)) $\Rightarrow$ $\neg$ P1.X)	
$\wedge\,\Box$	($\Theta_2 \wedge$ P2.X $\Rightarrow$	((V12 = open $\wedge$ V11 = closed $\wedge$ V15 = closed $\wedge$ heater = off) *Until* (B4 = empty (sol70C) $\wedge$ B5 = sol70C)))
$\wedge\,\Box$	((B4 = empty (sol70C) $\wedge$ B5 = sol70C) $\Rightarrow$ $\neg$ P2.X)	
$\wedge\,\Box$	($\Theta_3 \wedge$ P3.X $\Rightarrow$	((heater = on $\wedge$ V13 = open $\wedge$ V12 = closed $\wedge$ V15 = closed $\wedge$ V20 = closed) *Until* (B5 = Sol42H $\wedge$ B6 = water28H (water56H))))
$\wedge\,\Box$	((B5 = Sol42H $\wedge$ B6 = water28H (water56H)) $\Rightarrow$ $\neg$ P3.X)	
$\wedge\,\Box$	($\Theta_4 \wedge$ P4.X $\Rightarrow$	(V15 = open $\wedge$ V12 = closed $\wedge$ V18 = closed $\wedge$ heater = off) *Until* (B5 = empty $\wedge$ B7 = Sol42H (Sol84H))
$\wedge\,\Box$	((B5 = empty $\wedge$ B7 = Sol42H (Sol84H) $\Rightarrow$ $\neg$ P4.X)	
$\wedge\,\Box$	($\Theta_5 \wedge$ P5.X $\Rightarrow$	(V17 = open $\wedge$ V15 = closed $\wedge$ V18 = closed) *Until* (B7 = Sol42C (Sol84C)))
$\wedge\,\Box$	((B7 = Sol42C (Sol84C)) $\Rightarrow$ $\neg$ P5.X)	
$\wedge\,\Box$	($\Theta_6 \wedge$ P6.X $\Rightarrow$	(V29 = open $\wedge$ V13 = closed $\wedge$ heater = off $\wedge$ V20 = closed) *Until* (B6 = water28C (water56C)))
$\wedge\,\Box$	((B6 = water28C (water56C)) $\Rightarrow$ $\neg$ P6.X)	
$\wedge\,\Box$	($\Theta_7 \wedge$ P7.X $\Rightarrow$	(V18 = open $\wedge$ V23 = open $\wedge$ V22 = open $\wedge$ V1 = open $\wedge$ V3 = open $\wedge$ P1 = on $\wedge$ V15 = closed $\wedge$ V17 = closed $\wedge$ V8 = closed) *Until* (B7 = empty (Sol42C) $\wedge$ B1 = Sol42C (Sol84C)))
$\wedge\,\Box$	((B7 = empty (Sol42C) $\wedge$ B1 = Sol42C (Sol84C)) $\Rightarrow$ $\neg$ P7.X)	
$\wedge\,\Box$	($\Theta_8 \wedge$ P8.X	(V20 = open $\wedge$ V24 = open $\wedge$ V25 = open $\wedge$ V5 = open $\wedge$ V6 = open $\wedge$ P2 = on $\wedge$ V13 = closed $\wedge$ heater = off $\wedge$ V9 = closed) *Until* (B6 = empty (water28C) $\wedge$ B2 = water28C (water56C)))
$\wedge\,\Box$	((B6 = empty (water28C) $\wedge$ B2 = water28C (water 56C)) $\Rightarrow$ $\neg$ P8.X)	
$\wedge\,\Box$	($\Theta_9 \wedge$ P9.X $\Rightarrow$	((V8 = open $\wedge$ V3 = closed $\wedge$ V9 = closed $\wedge$ V11 = closed) *Until* (B1 = empty (Sol42C) $\wedge$ B3 = Sol42C)))
$\wedge\,\Box$	(B1 = empty (Sol42C) $\wedge$ B3 = Sol42C $\Rightarrow$ $\neg$ P9.X)	
$\wedge\,\Box$	($\Theta_{10} \wedge$ P10.X $\Rightarrow$	((V9 = open $\wedge$ V6 = closed $\wedge$ V8 = closed $\wedge$ V11 = closed) *Until* (B2 = empty (water28) $\wedge$ B3 = water28C)))
$\wedge\,\Box$	((B2 = empty (water28) $\wedge$ B3 = water28C) $\Rightarrow$ $\neg$ P10.X)	
$\wedge\,\Box$	($\Theta_{11} \wedge$ P11.X $\Rightarrow$	((V8 = open $\wedge$ Mixer = on $\wedge$ V3 = closed $\wedge$ V9 = closed $\wedge$ V11 = closed) *Until* (B1 = empty (Sol42C) $\wedge$ B3 = sol70C)))
$\wedge\,\Box$	((B1 = empty (Sol42C) $\wedge$ B3 = sol70C) $\Rightarrow$ $\neg$ P11.X)	
$\wedge\,\Box$	($\Theta_{12} \wedge$ P12.X $\Rightarrow$	((V9 = open $\wedge$ Mixer = on $\wedge$ V6 = closed $\wedge$ V8 = closed $\wedge$ V11 = closed) *Until* (B2 = empty (water28C) $\wedge$ B3 = sol70C)))
$\wedge\,\Box$	((B2 = empty (water28C) $\wedge$ B3 = sol70C) $\Rightarrow$ $\neg$ P12.X)	
$\wedge$	$\text{Liveness}_{(36)} \Rightarrow$	(Initial_Condition$_{(19)}$ $\Rightarrow$ Production_of_Batches$_{(20)}$)

Figure 4: Part 2 of the definitive theorem: the control specification and implication

chemical plants like in our case study it does not matter whether a heater is switched off a few milliseconds earlier or later. The plant is "slow" in comparison to the controller.

4.2 The implementation on a PLC

Given the above considerations a PLC program implementing our control specification can be derived in a straightforward way. The basic structure of the SFC control program can be found in figure 5. Each of the control processes specified above is implemented as a parallel component. The programs associated with steps (i.e. schemes) P1, ..., P12 are in detail in figure 6. There is no program associated with the wait-steps; the control process will simply idle there until the entry condition Θ_i holds. It will subsequently start repeating the execution of the control body program Pi cyclically until the postcondition result$i$ holds and then return to the waiting state.

In our case the schemes P1,..., P12 are particularly simple: they simply consist of loading a boolean value into the central register and then copying that value into the output memory locations corresponding to the actuators (valves, pumps, mixer, and heater) of the plant. The assembly-like instruction lists are preceded by so-called *action qualifiers*: action qualifier "P1" identifies those instructions that must be executed in the first execution cycle of the program, and qualifier "P0" marks those that must be carried out in the last cycle. This means that the "P0" instruction list is executed once when the corresponding postcondition result$i$ has become true. There exist several more action qualifiers (e.g. "N", for those instruction lists that must be executed in every cycle), which we do not need for our controller.

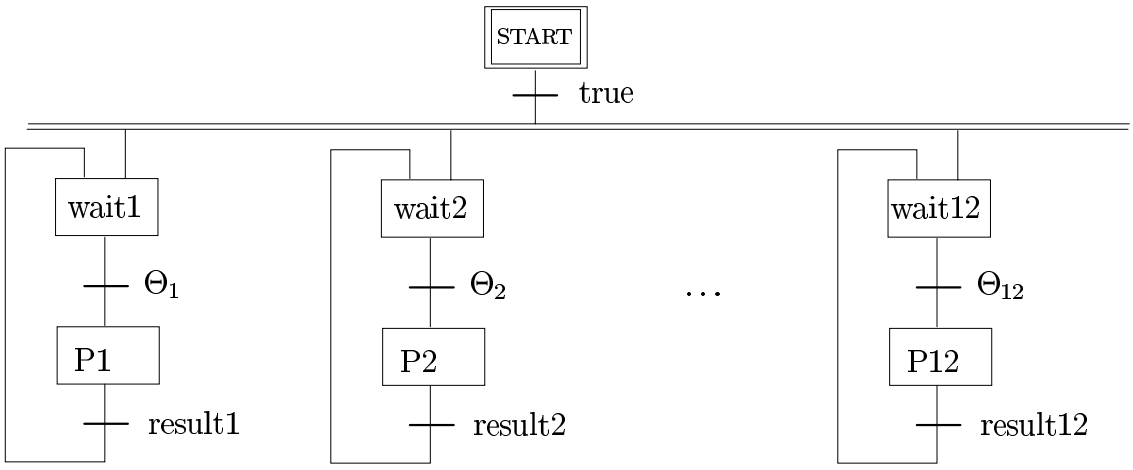

Figure 5: Structure of the Control Program in Sequential Function Charts

5 Verification of the controller

So far, we have developed the specification of a control program for the plant and derived a PLC implementation for it. The design theorem of Figures 3 and 4 has not been proven correct yet, however. A proof of this theorem will establish the correctness of the control specification, and implies that every PLC programmed to satisfy the control specification will interact with the plant in the desired way.

P1 :

P1	LD true ST V8
P0	LD false ST V8

P2 :

P1	LD true ST V9
P0	LD false ST V9

P3 :

P1	LD true ST V8 ST Mixer
P0	LD false ST V8 ST Mixer

P4 :

P1	LD true ST V9 ST Mixer
P0	LD false ST V9 ST Mixer

P5 :

P1	LD true ST V11
P0	LD false ST V11

P6 :

P1	LD true ST V12
P0	LD false ST V12

P7 :

P1	LD true ST Heater
P0	LD false ST Heater

P8 :

P1	LD true ST V15
P0	LD false ST V15

P9 :

P1	LD true ST V17
P0	LD false ST V17

P10:

P1	LD true ST V29
P0	LD false ST V29

P11:

P1	LD true ST V18 ST V23 ST V22 ST V1 ST V3 ST Pump1
P0	LD false ST V18 ST V23 ST V22 ST V1 ST V3 ST Pump1

P12:

P1	LD true ST V20 ST V24 ST V25 ST V5 ST V6 ST Pump2
P0	LD false ST V20 ST V24 ST V25 ST V5 ST V6 ST Pump2

Figure 6: Instruction List Programs for steps P1. ..., P12

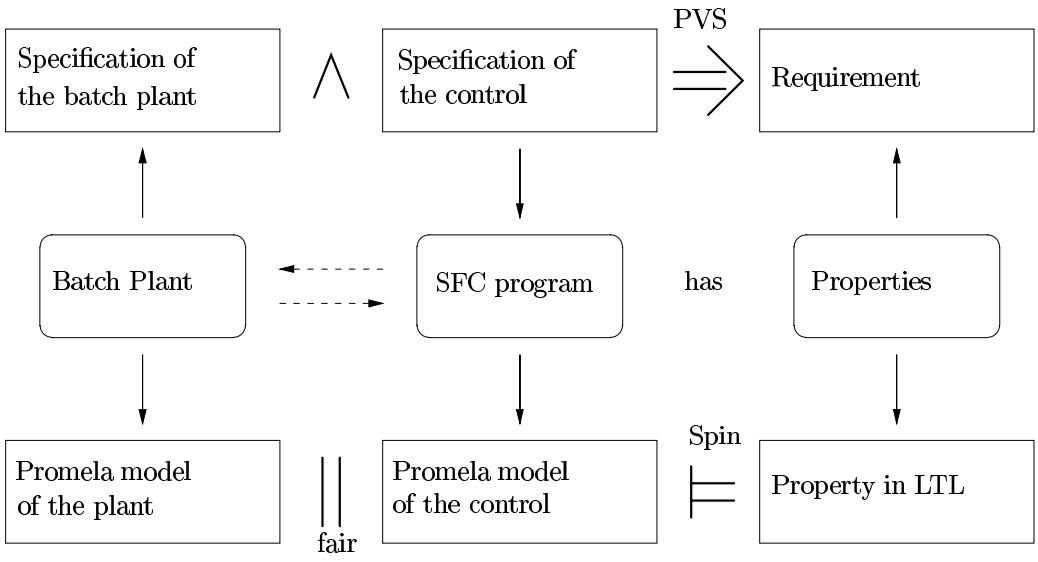

Figure 7: The two verification approaches

To handle the verification of the controller we have used two main tool-supported approaches, viz. theorem proving assisted by the automated theorem provers/proof assistants PVS [22], and model checking using the model-checking tool Spin [8]. The relation between the batch plant, the controller and these two verification approaches is depicted in Figure 7.

5.1 Verification by theorem proving

Verification by theorem proving using the proof assistant PVS[22] was our first option as we had developed and tested our initial specifications with the aid of PVS, as explained in section 2.3.

We successfully proved a simplified version of the design theorem, where only a single batch volume is produced. The PVS definitions of the specification formalism, our specifications, the simplified design theorem, and its PVS proof can be found in [26]. As an example we show one of the process step specifications in PVS notation:

```
Transport_from_B1_to_B3(t_transp: FUTURE): Specification =
  [] ( B#1=Sol42C /\ B#3=empty
       /\ ([]t_transp)(V#3=closed /\ V#9=closed /\ V#11=closed /\ V#8=open)
       => O(t_transp)(B#3=Sol42C /\ B#1=empty)
     )
```

Figure 8: The PVS model of transportation between B1 and B3

The parameter `t_transp` represents the real-time upper-bound on the transportation step that we already alluded to in section 3.3. In the case of PVS it turns out that a proof of bounded liveness is as easily obtained as that for general liveness, as we can provide uniform instantiations for the existentially quantified variables in the proof. In fact, theorem and proof do not depend on concrete time constants; with respect to production durations the theorem is generic. The proof basically follows the one batch moving through the plant and required little other handwork than

suggesting the correct instantiations for the production steps. The parametric real-time bounds did not cause any problems as PVS is good at handling symbolic arithmetic.

To complete the proof, we had to formalise the implicit domain knowledge that was not included in our theorem. The exclusion of different contents of the same container at the same point in time mentioned earlier in section 3.2 is a case in point. Or, that the contents of a container remains the same as long as all the valves connected to it are closed. The latter is a typical weakness of logic-based versus a model-based specification: one easily forgets to specify that things do not change as long as nothing happens. Such domain knowledge could, however, be added to the PVS-theory in the form of a number of generic "first principles". The number of valves in the plant did not cause any complication: there are about 30 valves, leading to a potential state space of 2^{30} only for representing the state of the valves. The reachable state space, however, is much smaller, and during the proof only reachable states had to be considered.

The preconditions, control specifications, and proof for the single batch version are considerably simpler than for the multiple batch cases allowed by Figure 4. The control specification basically boils down to a static schedule: a list of moments at which certain valves, etc. have to be opened and closed. No mutual exclusion mechanism is needed. The proof strategy we used for that single batch case was based on a careful case analysis of the production steps of the plant. This approach, where the proof follows the batch through the plant, cannot easily be generalized to the multiple-batch case, where the initialisation constraint alone allows for more than 8100 initial configurations. Any naive attempt to prove the theorem leads to a combinatorial explosion of proof obligations.

These facts motivated our second approach to verification using model checkers, which we present below. Still, the verification by theorem proving of the simplified single batch case proved very useful as going explicitly through every proof step gave good insight in how control and plant interact. It also gave us strong indications that the plant specification was adequate.

After having completed the proof we can, of course, conclude only that the control *specification* is correct (for the single batch case). It remains to show that also the given implementation satisfies the specification. For that purpose a semantics of SFC programs in terms of e.g. the modal language used is necessary and an implication between implementation and specification has to be shown. We have not pursued this further formalisation. Instead, we have tried to make the ultimate formulation of the control specification fairly concrete, so that an implementation could be obtained from it more or less directly. This close connection permits us to see the final specification as a *logical model* of the control program. The design theorem then does state the correctness w.r.t. this model.

5.2 Verification by model-checking

In model-checking a verification model of the derived implementation is constructed directly. Such models are often closer to the operational reality of the implementation than a logical specification, and in this sense yield a better verification of the implementation. A related disadvantage is the risk of introducing unnecessary operational detail, leading to a less abstract model, and therefore less general result.

The main reason to switch to model checking, however, was the difficulty to generalise PVS proof to the general, multiple batch case, as mentioned above. It turned out to be feasible to run the model checker sequentially on our model initialised with material for 0 up to 8 batches (including the intermediate different possibilities for half batches; 30 runs in total). In order to avoid the explosion of the more than 8100 possible initial configurations allowed by (19), we considered only configurations filling the plant "from the top", i.e. filling tanks in the order B1,...,B7. The other

initializations are reachable from these by normal operation of the plant. As satisfaction of the property that we checked (see below) for our initial configurations implies its satisfaction for all reachable configurations this is sufficient. Using simulations of our model we satisfied ourselves that our model did include the required normal operation steps (here, model checking would run into the same combinatorial explosion).

To do our model-checking we used the model-checker Spin and its related modelling language Promela [8]. Given the fact that in our design and verification we can abstract from many of the continuous features of the hybrid system of the plant this is a reasonable choice. Existing hybrid or real-time model checkers are to be preferred only if a direct treatment of the continous variables is required; for non-hybrid, non-timed systems Spin has a strong performance record. Also, it is interesting to see to what extent we can deal with the verification of this class of systems using standard theory and tools.

There are two basic principles that allow us to deal with a real-time system while abstracting from time (see also [20] for a more general account in the context of PLCs):

1. The control program works independently of the time that the production steps take. Therefore, in the model each of the production steps S1, ..., S12 may take some unspecified time: if activated (e.g. by opening a valve) it goes to an undefined state that it eventually will leave to reach the final state where its postcondition holds. By this way of modeling every timing behaviour of the plant is subsumed, including the real one. If we can prove correctness for this general case, then correctness of the special case follows.

2. We can assume the excution speed of the control program to be much faster than that of the plant processes. The PLC program is executed in a continuous cycle (scan cycle), where each program execution is preceeded by a polling of input (sensor data) and writing output to the output points (switching on and off actuators). The execution time of a scan cycle is in the range of a few milliseconds. For some applications the timing behaviour in this range is relevant, e.g. for machine control. For for our chemical plant it is not relevant: it does not really matter whether a valve closes 10 ms earlier or later. This has two important implications:

   - we can abstract from the scan cycle time and assume that scan cycles are executed instantaneously.
   - we can assume that the plant is continuously scanned so that state changes are detected without (significant) delay.

In our model of the combined behaviour of the plant and the control program we must make sure that the continuous execution of the control program does not cause a starvation of the plant processes. This is taken care of my allowing only fair executions in Spin of our Promela model: in each execution no active process may be ignored indefinitely. We must be careful, however, not to lose the other important property, viz. that each state change of the plant is detected "immediately". Our model takes care of this by forcing a control program execution after each potential state change of the plant.

The full Promela model of this case study can be found in [26]. Here we present two excerpts, one of the plant model and one of the control program model, to illustrate its main features. Figure 9 contains the Promela process that models the transportation of solutions from container B1 to B3. It models the combined behaviour underlying steps S1 and S3.

The model consists of a do-loop that continuously tries to start the transportation of a unit of salt solution from B1 to B3 (corresponding to steps S1 and S3 of the specification). If the right

```
proctype B1toB3()
{   do
    ::  atomic{ (cycle==0 && B1!=cempty && v8) ->
                if
                ::  (B1==sol42C) -> B1=undef1
                ::  (B1==sol84C) -> B1=undef2
                ::  else -> error
                fi ;
                if
                ::  (B3==cempty) -> B3=undef1
                ::  (B3==water28C && mix) -> B3=undef2
                ::  else -> error
                fi ;
                cycle=1
        } ;
        assert(v8 && (B3!=undef2 || mix)) ;
        atomic{ (cycle==0 && v8) ->
                if
                ::  (B1==undef1) -> B1=cempty
                ::  (B1==undef2) -> B1=sol42C
                ::  else -> error
                fi ;
                if
                ::  (B3==undef1) -> B3=sol42C
                ::  (B3==undef2 && mix) -> B3=sol70C
                ::  else -> error
                fi ;
                cycle=1
        }
    od
}
```

Figure 9: The Promela model of transportation between B1 and B3

conditions are fulfilled control will enter the body of the loop, and will mark the beginning of the transport step by instantaneously (using the Promela `atomic` construct) changing the contents of both containers to undefined transitional states. At some later moment it will execute the second part of the body, instantaneously changing the transitional states into corresponding terminal states, modelling the end of the transport step. Between these two atomic constructs there is an assert statement that checks that at any point during the transport step the asserted statement holds. The following points may help the reader to understand the Promela model better. For a tutorial on Promela we refer to [8, 23].

- The `cycle` variable is a global flag that forces the excution of a scan cycle after the execution of each atomic step in the plant (flag is raised at the end of each such atomic step). After the execution of a scan cycle (also modelled as an atomic process, see below) the flag is lowered. Each atomic step in the plant is guarded by the test `cycle==0`.
- The Promela model combines steps in the plant that involve the same set of containers into one process. This reduces the number of processes that must be scheduled fairly.
- The Promela model of the plant models the transportation steps from a "physical" attitude and imposes fewer conditions for a transportation to take place that the formal plant specification in the corresponding steps. E.g. for transportation from B1 to B3 to take place it is only required that B1 is not empty and valve V8 is open.
- To compensate for this all illegal and unwanted states of the plant are explicitly modelled as error states (`error` is defined as `assert(false)`) whose reachability can be checked. This approach gives us more information about the robustness of our controller.

The Promela process that models the control program is listed in Figure 10. This is a straightforward translation of the PLC program (and its specification).

The do loop of `Control` repeatedly executes an atomic scan cycle, in which the processes P1,...,P12 are scheduled sequentially. To deal with the symmetric subcases of each step (i.e. the one written between brackets in the specification) we need a second loop counter `j` next to the main counter `i` (because P11 and P12 in fact have 4 subcases P11 is covered by `i`$\in \{11, 12\}$ and `j`$\in \{1, 2\}$, and P12 by `i`$\in \{13, 14\}$ and `j`$\in \{1, 2\}$). Modulo these small adaptations the `theta(i,j)` correspond to the Θ-predicates of the specification and the program, the `result(i,j)` correspond to the result conditions of the PLC program (given as explicit predicates in the specification), and `PB1(i)` and `PB2(i)` correspond to the code of the P1 part, and the P0 part, of the Pi steps of the PLC program, respectively. The variables `px[i]` correspond to the Pi.X activity predicates of the specification. Note that at the end of each scan cycle the global flag `cycle` is lowered, as required.

The property language for the Spin model-checker is linear temporal logic (LTL). The requirement (20) of our property language can also be read as an LTL formula, as does not contain any of the time-decorated modal operators. Having moved from the original continuous time domain to the discrete, non-dense time domain of the Promela model we have also lost the unintended Zeno-behaviours that were allowed by the continuous domain.

After initial simulations and model checking runs had been used to remove small (mainly syntactic) mistakes from our model, the model was systematically checked for property (20) for the 30 initializations with different batch volumes described above. No errors were reported, except for initializations with batch volumes 0, 0.5, 7.5 and 8, as should be the case. The model checking was done using Spin version 3.3.7 on a SUN Enterprise E3500-server (6 SPARC cpus with 3.0 GB main memory). The model checking was run in exhaustive state space search mode with fair scheduling. The error states reported unreachable in all runs. The shortest runs were completed in the order

```
proctype Control()
{ int i,j ;
    do
    ::  atomic{ i=1 ; j=1 ;
            do
            ::  (i<15) ->
                if
                ::  (theta(i,j) && !px[procnr(i)]) -> PB1(i)
                ::  (result(i,j) && px[procnr(i)]) -> PB0(i)
                ::  else -> skip
                fi ;
                if
                ::  (j==1) -> j=2
                ::  (j==2) -> j=1 ; i=i+1
                fi
            ::  (i==15) -> goto endcycle
            od ;
            endcycle:  cycle=0
        }
    od
}
```

Figure 10: The Promela model of the control process

of seconds (user time) and consumed in the order of 20MB memory; the longest run required in the order of 40 minutes and 100MB.

6 Related work

Being a case study of the VHS project, the batch plant control problem has also been studied by a number of other authors. In this section we will compare our work to theirs. We also compare our method of incremental design to some other approaches in the area of real-time reactive systems.

The use of formal methods for the specification and analysis of chemical process control has already some tradition. A related case study to which of a variety of techniques from both control theory and computer science has been applied, is the so-called "evaporator problem" [12]. This problem is in fact a subproblem of our case study, pertaining only to the use of the evaporator. Its formulation requires a more quantitative approach to the dynamics of the plant dynamics, however. Besides ourselves, a number of others from the VHS project have reported results on the batch plant case study that is the subject of this work [25]. In addition to the evaporator problem [9], they adress the problem of optimal scheduling of the plant processes [21] and the correctness of given SFC control code [15, 19]:

- Niebert and Yovine [21] assume the correctness of the lower level PLC routines and consider the problem of (time) optimal scheduling of these routines. Their methodology is based on a timed automaton model with shared variables. The reformulate the scheduling problem as a reachability problem and apply two model checking tools OpenKronos and SMI to it.
- Kristoffersen et al. [15] model given control routines, a coordinator process and the plant as timed automata, which as subsequently analysed with the tool Uppaal. Like us, they

discretise the continuous tank volumes, but on a finer scale (integers). The main verification effort is dedicated to safety requirements (e.g. all pumps and valves are closed again when an SFC routine terminates).

- Lukoschus [19] also addresses the correctness of givcen SFC code, but does so in the untimed model of Discrete Condition Event Systems (DCES). Again the volume of the tanks is discretised, abstracting it to the states *empty* and *full*. Also here a large number of safety properties could be proven.

As we made the design of the SFC code a part of our problem formulation, our approach is different from the latter two. Because many of the safety features were almost explicitly constructed properties of our design, we have concentrated on the verification of the infinite production of batches. As this is not a reachability property, this cannot be verified (directly) with Uppaal. In this paper we have not addressed the optimality of our controller design. In a related publication [2], however, we show how our Promela model can be extended to deal with this problem. Our setup is a bit different than that in [21], but our main findings are consistent with their results.

We have used a design method based on *incremental design*. Formalisations of such incremental approaches to the case of hybrid (in fact: real-time) systems can also be found in [16, 3]. These are more formal treatments in the sense that they are built around formal refinement (or superposition) steps in which proven correctness properties are preserved in some apropriate sense by the application of such steps. Although our approach can be reformulated in such a way, we have not distributed our correctness proofs over the different levels of abstraction, we have only distributed the specification of the design over such levels supported by informal arguments of their correctness. This gave us more flexibility in (ab)using the formal framework to analyse our design decisions (e.g. by allowing invalid versions of the design theorem), but also left us with the considerable task of formally verifying the final version of the specification theorem at the end. It would be interesting to analyse which approach would be the better one under what circumstances. We believe that ours can be beneficial in the *context of discovery* (see section 2.1). Another example of design using a formal approach and allowing invalid intermediate stages can be found in [5], whose more formal reconstruction can be found in [4].

7 Conclusion

We have shown that incremental design using a temporal logical formalism and a pragmatic refinement strategy that allows for invalid intermediate designs, can be used to specify the hybrid system of the batch plant case study in a clear and structured way. We have demonstrated how this structure can be used to derive a PLC controller for the plant. The approach has proven particularly useful to identify and analyse the parallelism of the basic processes of such systems, which is not always obvious from the start.

We believe that our approach and results are applicable to plant control in general (and perhaps even wider classes of hybrid systems), as the structuring principles appear quite generic. Our treatment of the real-time aspects requires the operation of the plant to be much slower than the speed of controlling processes. Our discretisation of continuous variables like temperature and volumes is allowed when a (small) finite number of values suffices for the control. These conditions are certainly reasonable when production takes place in the form of fixed sized batches. Our feeling is further substantiated by the fact that a similar, smaller case study involving the PLC control of a LEGO plant sorting blocks resulted in a controller architecture quite comparable to that of figure 5 [14]: competing, concurrent programs for each basic plant process executed in lock step in each scan cycle.

For the verification of our design we used both theorem proving with PVS, and model checking with Spin/Promela. At first sight the theorem proving approach seemed more attractive because it does not suffer from the state explosion problem. Even in this simple plant the combination of all the valves alone account for a potential state space of 2^{30} different states. In the absence of a more general proof strategy for this type of problem, however, we could only work with an approach that relied on a rather elaborate case analysis. In this manner we could analyse the simpler case of a single batch plant, but this did not scale up to cases of multiple batches because of an unmanageable number of generated proof obligations. The development of a more general PVS strategy that would circumvent this problem, remains an interesting problem.

A strong point of the PVS verification was the ease with which we could prove bounded liveness properties for the plant by providing proper instantiations for generic time bounds in the proof. In fact, we expect that such instantiations can be automated by straightforward PVS proof strategies.

The model-checking approach did work for the multiple batch cases, as it turned out that the reachable part state space was substantially smaller than the theoretical maximum. In fact, we could do an exhaustive state space search in all cases. The state space requirements were not exceptional, but interestingly enough the required *depth* of the search trees was quite big (up to 500000 steps).

Spin and Promela were not devised for hybrid systems, but given our ultimate discretisation of the problem, could handle the challenge quite well. In a related publication the first two authors have shown how the models can be extended to derive even time-optimal schedules with the aid of these tools using the technique of *variable time advance procedures* [2]. It will be interesting to compare such results with those obtained with the aid of real-time and hybrid model checking tools like Uppaal [17], KRONOS [1] or HYTECH [7].

In spite of our results we have, of course, dealt with only a part of the problems of the verification of PLC applications. We are also in the course of actually implementing the derived PLC controller, which brings a number of essential, practical problems to light. One important difficulty of the case study is that the implementation is not just a program, but a program distributed over some 40 different files. Another real source of errors is that (by the nature of PLCs) hardware adresses have to be used. In combination with space restrictions (more variables cost more money) variables have to be packed as efficiently as possible into the available space. This does not always lead to the clearest structure of the programs. In our case variable declarations are spread over about 12 files, because only a certain number of variables can be declared per file, etc., etc. For many PLCs programming environments these facts lead to additional sources of errors. To successfully verify PLC applications simpler possibilities for structuring programs are necessary. At any rate it is clear that any verification effort, no matter how successful, must always be supplemented by systematic testing of the implementation to detect such implementation-generated errors. The developed specifications can be used as a source for the generation of such tests.

References

[1] M. Bozga, C. Daws, O. Maler, A. Oliveiro, S. Tripakis, and S. Tripakis. Kronos: A model-checking tool for real-time systems. In *Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer-Verlag, 1998.

[2] E. Brinksma and A. Mader. Verification and optimization of a plc control schedule. In *SPIN Model Checking and Software Verification*, volume 1885 of *Lecture Notes in Computer Science*, 2000.

[3] A. Cau and W.-P. de Roever. Using relative refinement for fault tolerance. In *FME'93: Industrial Strength Formal Methods*, volume 670 of *Lecture Notes in Computer Science*, pages 19 – 41, 1993.

[4] A. Cau, R. Kuiper, and W.-P. de Roever. Formalising dijkstra's development strategy within stark's formalism. In *Proceedings Fifth BCS-FACS Refinement Workshop*, 1992.

[5] E.W. Dijkstra. A tutorial on the split binary semaphore. *http://www.cs.utexas.edu/users/EWD/index07xx.html*.

[6] A. Fehnker. Scheduling a steel plant with timed automata. In *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*. IEEE Computer Society Press, 1999.

[7] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. In *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–463. Springer-Verlag, 1997.

[8] G.J. Holzmann. The model cheker SPIN. *IEEE TRansactions on Software Engineering*, 23(5):279–295, May 1997.

[9] Ralf Huuck. Verifying timing aspects of VHS case study 1. Technical Report (number to be assigned), Christian-Albrechts-Universität zu Kiel, 1999.

[10] International Electrotechnical Commission. *IEC International Standard 1131-3, Programmable Controllers, Part 3, Programming Languages*, 1993.

[11] S. Kowalewski. Description of case study CS1 "Experimental Batch Plant". Draft. University of Dortmund, Germany, July 1998.

[12] Stefan Kowalewski and Olaf Stursberg. The batch evaporator: A benchmark example for safety analysis of processing systems under logic control. In *Proceedings 4th Workshop on Discrete Event Systems (WODES)*, pages 302–307. IEE, London, 1998.

[13] R. Koymans. Specifying message-passing and real-time systems with real-time temporal logic. Technical report, Eindhoven University of Technology, Dept. of Mathematics and Computing Science, Eindhoven, The Netherlands, 1987.

[14] J. Kratz. A case study in PLC control. Master's thesis, University of Nijmegen, 1999. see also *http://www.cs.kun.nl/~ mader/LEGO/*.

[15] K. Kristoffersen, K. Larsen, P. Pettersson, and C. Weise. VHS Case Study 1 - Experimental Batch Plant using UPPAAL. BRICS, University of Aalborg, Denmark, May 1999.

[16] R. Kurki-Suonio. Stepwise design of real-time systems. *IEEE TRansactions on Software Engineering*, 19(1), January 1969.

[17] K.G. Larsen, P. Petterson, and W. Yi. UPPAAL in a nushell. *Int. Journal of Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[18] R. W. Lewis. *Programming Industrial Control Systems Using IEC 1131-3*. Control Engineering Series. The Institution of Electrical Engineers, London, 1999.

[19] Ben Lukoschus. An abstract model of VHS case study 1 (experimental batch plant). Technical Report (number to be assigned), Christian-Albrechts-Universität zu Kiel, 1999.

[20] A. Mader. A classification of PLC models and applications. submitted to WODES, 2000.

[21] Peter Niebert and Sergio Yovine. Computing optimal operation schemes for multi batch operation of chemical plants. VHS deliverable, May 1999. draft.

[22] PVS homepage. *http://pvs.csl.sri.com/*.

[23] SPIN homepage. *http://wwwnetlib.bell-labs.com/netlib/spin/whatispin.html*.

[24] VHS project homepage. *http://www-verimag.imag.fr/VHS/main.html*.

[25] VHS project deliverable cs.1.1: Report on VHS case study 1. *http://www-verimag.imag.fr/VHS/main.html*.

[26] VHS: Case study 1 sources. *http://www.cs.kun.nl/~ mader/vhs/cs1.html*.

[27] H. Wupper and A. Mader. System design as a creative mathematical activity. Technical Report CSI-R9919, University of Nijmegen, Computing Science Institute, 1999. *http://www.cs.kun.nl/csi/reports/info/CSI-R9919.html*.

[28] Hanno Wupper. Quantified modal algebra: Quantified modal operators and their use in specification. Technical Report CSI-R9730, University of Nijmegen, Computing Science Institute, December 1997.